UNIVERSIDAD AUTÓNOMA DE ZACATECAS

DISEÑO DE IP CORES DE CIFRADO APLICADO A TELECOMUNICACIONES

Laura García Luciano

Tesis de Maestría

presentada a la Maestría y Doctorado en Ingeniería y Tecnología Aplicada

de acuerdo a los requerimientos de la Universidad para obtener el título de

MAESTRA EN INGENIERÍA Y TECNOLOGÍA APLICADA

Directores de tesis:

M. en I. T. C. Salvador Ibarra Delgado y Dr. Remberto Sandoval Aréchiga

POSGRADO CON LA INDUSTRIA EN INGENIERÍA Y TECNOLOGÍA APLICADA

Zacatecas, Zac., 4 de Octubre del 2018

AUTORIZACION DE IMPRESIÓN DE TESIS

Laura García Luciano

P R E S E N T E

La dirección de la Unidad Académica de Ingeniería Eléctrica le notifica a usted que la Comisión Revisora de su Documento de tesis de maestría, integrada por los profesores M.I.T.C. Salvador Ibarra Delgado y el Dr. Remberto Sandoval Arechiga, Dr. Jorge Flores Troncoso. Dr. Viktor Iván Rodríguez Abdala y el Mtro. Gerardo Ornelas Vargas, ha concluido la revisión del mismo y ha dado la aprobación para su respectiva presentación.

Por lo anterior, se le autoriza a usted la impresión definitiva de su documento de tesis para la respectiva defensa en el Examen Profesional, a presentarse el día 04 de Octubre del 2018.

Atentamente

Zacatecas, Zac., 24 de Agosto de 2018

DR. JORGE DE LA TORRE Y RAMOS
DIRECTOR DE LA
UNIDAD ACADEMICA DE INGENIERÍA ELECTRICA

Av. Ramón López Velarde No. 801. CP 98060.
Zacatecas, Zac. Tél: (492) 92 2 0827
piitec@uaz.edu.mx

Agradecimientos

Éste trabajo no hubiera podido ser realizado sin la beca de CONACYT para el Programa de Maestría en Ingeniería y Tecnología Aplicada.

Agradezco al Centro de Investigación, Innovación y Desarrollo en TElecomunicaciones por darme la oportunidad de trabajar en uno de sus muchos proyectos en Telecomunicaciones.

Agradezco a mi mentor y asesor de tésis el M. en I.T.C. Salvador Ibarra Delgado por guiarme, por compartirme su conocimiento y por su paciencia a lo largo de la elaboración del trabajo. Del mismo modo, estoy muy agradecida con el Dr. Manuel Hernández Calviño por compartirme su conocimiento de la manera más clara y en repetidas ocasiones sin desesperarse.

También quiero hacer un especial agradecimiento a quien me enseñara sobre las matemáticas en el cifrado de datos, quien sin conocerme me brindó todo su apoyo, Dr. Homero Gallegos Ruiz. Así mismo, agradezco al Dr. Remberto Sandoval Aréchiga por asesorarme en el proceso de revisión que mejoró en gran medida ésta tesis.

Agradezco a Juve por darme su apoyo total y amor incondicional en todo el camino.

Agradezco a mis padres Tere y Benito y a mi hermana Moni por todo su cariño y sobre todo por su entendimiento en éste largo proceso.

Finalmente, agradezco a familiares y amigos que con sus mensajes y pláticas me mantenían con la mente tranquila.

RESUMEN

En la actualidad, se tiene un uso masivo de las telecomunicaciones y la información que se transmite es en su mayoría sensible. Existen desarrollos de elementos que hacen que dicha información sea ilegible a la vista de terceros no autorizados, sin embargo, no son reconfigurables y no es posible realizarle mejoras que eviten riesgos de privacidad.

Este documento comprende el desarollo de un IP Core de cifrado AES-128/256 implementado en un Dispositivo Lógico Programable, que puede ser parte de un Sistema de Telecomunicaciones. El cifrador AES se conforma de un IP Core que cifra los datos y un IP Core que recupera los datos originales. Éstos IP Cores se desarrollaron de forma que sean reconfigurables por medio del software del sistema embebido en el que están contenidos, así como reutilizables en otros posibles sistemas digitales con otras aplicaciones debido a que cuentan con un protocolo estándar llamado AXI4-Stream que les permite comunicarse con otros sistemas que utilicen el mismo protocolo.

Primero, se realizó un estudio del estado de la cuestión de los últimos cuatro años, profundizando particularmente en algoritmos de cifrado sobre FPGAs. Seguido de la comprensión de los conceptos que giran alrededor de un cifrador AES y el estudio de los diferentes elementos que son necesarios para la implementación hardware del mismo. AES cifra bloques de 128-bit cada vez, y utiliza una misma clave de 128/192/256-bit para cifrar y para descifrar, por lo que recibe el nombre de cifrador simétrico. Dicho cifrado consiste en un número de rondas que se aplican al bloque de datos de entrada, y en la última ronda el bloque de datos resultante es el dato cifrado o también conocido como criptograma. El diseño de la arquitectura hardware del estándar de cifrado AES, se describió y se simuló en Verilog tanto para el IP Core de Cifrado como para el IP Core de Descifrado. Además, les fue añadido un protocolo de comuncicación denominado AXI4-Stream que les permite comunicarse con cualquier módulo hardware que cuente con la misma interfaz.

La implementación del sistema fue realizado utilizando la tarjeta de desarrollo Zedboard cuyo elemento principal es el Zynq . El desarrollo constó de dos elementos principales. El primero, una plataforma de hardware en la que se incluyen los dos IP Cores. Y el segundo, una plataforma de

software capaz de controlar las entradas de datos al sistema, por medio de una hiper terminal. Con lo que se pudo verificar el cifrado AES-128 y descifrado AES-128 (ambos AXI4-Stream) de bloques de 128-bit de datos. La verificación del funcionamiento de los bloques hardware diseñados, fue contrastada con los vectores de prueba diseñados para este efecto por el Instituto Nacional de Estándares y Tecnología (NIST) [1].

Contenido General

Pag.

Lista de figuras

Figura Pag.

Figura Pag.

Lista de tablas

Capítulo 1

Introducción

Las telecomunicaciones juegan un rol central en la sociedad puesto que proporcionan una base tecnológica para las comunicaciones sociales, y es de gran importancia que a reserva del emisor y receptor de la información, terceros no tengan acceso a ella. Para que la información que es transmitida sea privada, es decir, que sólo los dueños de la información puedan accesarla de forma entendible, es necesario utilizar algoritmos de cifrado.

Un algoritmo de cifrado le permite al emisor y al receptor mantener la información oculta y dependiendo del entorno de la información que se quiere transmitir, se escoge dicho algoritmo de una lista que primeramente cuenta con dos clasificaciones: la simetría de sus claves y el modo en que será alimentado el cifrador con la información (bit a bit ó bloques de bits). Después que se tiene un algoritmo de cifrado conveniente, se tiene que implementar ya sea en software o en hardware, y ésta desición depende del sistema en que se quiera colocar el algoritmo de cifrado. En el entorno donde algunos productos, como es el caso de esta tesis, quedan embebidos en un circuito electrónico, y donde también existe información sensible dentro de ellos, un algoritmo de cifrado para hardware es lo ideal.

Un ejemplo de sistema donde se puede utilizar algoritmos de cifrado es un radiomódem, el cual es un dispositivo electrónico que transfiere información crítica de forma inalámbrica. En la mayoría de los casos, los radiomódems forman parte de una red de radio privada. Éste dispositivo requiere en primer lugar, que su información esté oculta de terceros y su alta tasa de transferencia no se vea afectada significativamente, por lo que con la implementación hardware de un algoritmo

de cifrado que cumpla con los requerimientos antes mencionados, queda resuelta ésta primera necesidad. Una segunda necesidad, y no menos importante, es que el módulo de cifrado pueda interactuar con aquellos que componen al radiomódem, los cuales son explicados a detalle en la Sección 2.1, y ello se resuelve utilizando un estándar de comunicación entre módulos hardware.

1.1 Planteamiento del Problema

1.1.1 Antecedentes

La información que se transmite en distintos sistemas de telecomunicaciones comunmente es confidencial o sensible. La confidencialidad de dicha información puede ser comprometida cuando se accesa a ella sin autorización, cuando se copia o cuando se esparce por terceras partes, y como consecuencia se pierde la privacidad de la misma, y en casos extremos, se podría dañar la reputación del propietario de la información. Por ejemplo, en una transacción bancaria electrónica fallida se puede perder fácilmente la confianza de los usuarios y eso generaría pérdidas para el banco, etc.[2]. Por lo tanto,la información vulnerada genera consecuencias que van desde las personales hasta las económicas que recaen en los propietarios de la misma.

Para poder darse cuenta del valor de la información, el Reporte Anual de Cisco 2017 [3] menciona el impacto general sobre las compañías que no tienen un sistema de seguridad: el 22% pierde clientes, el 23% pierde oportunidades y el 29% pierde ingresos. Y aquellas compañías que si cuentan con algún sistema de seguridad hicieron caso omiso al 44% de las alertas emitidas por su sistema, y sólo el 49% de las compañías pudo manejar alguna violación de seguridad. En la Figura 1.1 obtenida del mismo reporte, muestra que sólo el 47% de los distintos sectores en México son conscientes de lo vulnerable que es la información que administran.

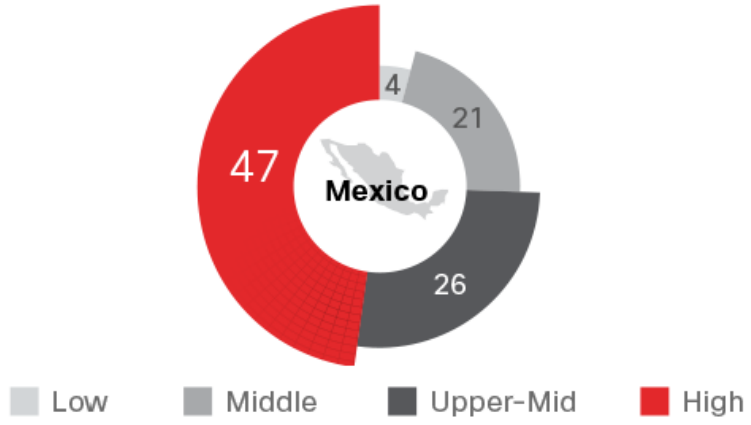


Figura 1.1: Madurez de la Seguridad en México[1]

[1]Gráfica tomada del Reporte Anual de Gráficos de Ciberseguridad Cisco 2017 Figura 67. [5]

Entonces, la confiabilidad en las telecomunicaciones es un área de especial preocupación para los proveedores de servicios y para sus usuarios, por lo que el emisor, el canal y el receptor deben estar protegidos de actividad maliciosa. Y para que la información que viaja a través de un Sistema de Telecomunicaciones sea privada, es necesario que los códigos binarios que representan esa información sean transformados, de modo que son transpuestos para que cuando exista una intercepción no autorizada se produzca una secuencia indescifrable de caracteres que sólo el receptor puede descifrar pues cuenta con el decodificador que restaura la información. Éste proceso de desordenar, transmitir y reordenar el mensaje es conocido como "cifrado".

El cifrado en las telecomunicaciones no debe ser uniforme ni universal, es decir, se debe cifrar según las necesidades y ninguna otra entidad debe contar con el conocimiento de como se lleva a cabo el ocultamiento de la información para evitar que el acceso a ella resulte fácil.

Hablando específicamente de México, se sabe que en el tema de cifrado hardware, la mayoría de los productos son de origen extranjero[2]. Y sin embargo, se encuentran esfuerzos de desarrollos hardware como en la compañía llamada *CEGA Security* que elabora módulos criptográficos (Hardware Security Module) que comunmente se utilizan para proteger la información de una nube y de resguardar claves criptográficas. Dichos módulos hardware no cuentan con la disponibilidad de ser rediseñados digitalmente, y como consecuencia, no pueden ser reutilizados en otras aplicaciones implementadas en sistemas digitales. Entonces, se sigue teniendo la necesidad de desarrollar un bloque hardware que tenga la función de cifrar y que sea reconfigurable.

Por lo que es necesario en México el desarrollo de IP Cores[3] de Cifrado con algoritmos estándar y con configuraciones propias. Éstos IP Cores también deben poder incorporarse a un System on Chip (SoC)[4]donde se lleve a cabo un proceso en el que se requiera que los datos sean privados.

Es común el uso de dispositivos lógicos programables como los FPGAs para desarrollar IP Cores de Cifrado, debido a que su crecimiento en capacidad a lo largo del tiempo y a su bajo costo en comparación con circuitos como los ASICs (Aplication-Specific Integrated Circuits). Entre sus

---

[2]Cuando se hace una búsqueda en internet sobre productos de cifrado hardware, los resultados son en su gran mayoría extranjeros, por ejemplo http://dornerworks.com/fpga-ip

[3]Bloque lógico funcional de Propiedad Intelectual que puede ser implementado en FPGAs o formar parte de un ASIC.

[4]Chip que tiene al menos un microcontrolador programable y una unidad de memoria.

ventajas permiten describir algoritmos en un dispositivo que es de naturaleza programable y que cuenta con grandes ventajas como un rápido tiempo de comercialización, la ausencia de costos futuros por la fabricación y que se trata de un silicio probado para diseño [6].

La aparición de los FPGAs se remonta a 1984, al inicio la capacidad de los FPGAs crecía en un factor de 10,000 y la velocidad en un factor de 100 [9], los ASICs aún tenían el terreno ganado. Inicialmente, la principal desventaja de un ASIC eran los circuitos que después de ser producidos se encontraban errores de diseño que consecuentemente hacía perder dinero.

Ahora, el costo y energía por operación han disminuído por encima de un factor de 1000 gracias a los efectos cuantitativos de la ley de Moore, efectos que también han cambiado la arquitectura de los FPGAs, sus aplicaciones y las herramientas usadas, poniendo a los FPGAs como líderes en diseño electrónico.

Uno de éstos cambios y que es de gran importancia mencionar, es la capacidad de un circuito de integrar elementos tales como unidades de procesamiento, unidades lógicas, multiplicadores, memoria, o simplemente módulos funcionales hardware como los IP Cores. Dicho circuito es comunmente conocido como SoC, que ya se ha mencionado en párrafos anteriores.

Los elementos integrados en un SoC deben poder comunicarse entre sí de un modo estándar, y es por ello que se utilizan protocolos de comunicación estándar para que esto sea posible.

La tarjeta de desarrollo *ZedBoard*™ en la que se implementan los IP Cores de Cifrado y Descifrado, cuenta con un SoC *Zynq7Z020* que integra un sistema de procesamiento y lógica programable. Así mismo, cada IP Core que se implementa en ésta tarjeta tiene una interfaz estándar AXI, por que independientemente de la tarea que tenga asignada un IP Core, parte de su funcionalidad es el poder enviar los datos procesados al IP Core que le corresponda a través de un protocolo de comunicación. Un IP Core de Cifrado AXI se convierte en una pieza que fácilmente pudiera agregarse a un sistema AXI que requiera de sus capacidades. Así como también, la reconfiguración del mismo.

1.1.2 Justificación

En México el desarrollo de núcleos de Propiedad Intelectual es aún limitado, y en el tema específico de IP Cores para cifrado de datos que puedan ser integrados a un SoC, ésta limitación es aún más evidente. México es un país altamente dependiente de tecnologías y servicios extranjeros, lo cual puede representar un riesgo para la seguridad del país, por lo que es importante comenzar a desarrollar los elementos que abarcan la seguridad en los Sistemas de Telecomunicaciones, uno de ellos son los IP Cores de Cifrado que contengan características propias y que sean desconocidas por otros países.

Al ser este proyecto un desarrollo cuyo elemento principal es desarrollado en el país, se tiene la posibilidad de contribuir a disminuir la dependencia tecnológica en términos de cifrado de datos. Además, se puede tener la posibilidad de incrementar las capacidades del IP Core de Cifrado que puede ser enriquecido en seguridad al cambiar su modo de operación (CBC, CFB, OFB, CTR, etc. [10]) dependiendo de la aplicación del Sistema de Comunicaciones. De ese modo, se cuenta con el potencial de un IP Core de Cifrado sin restricción de uso, como los que se encuentran en el mercado que en su mayoría son tecnologías de otros países.

1.1.3 Identificación del problema

México es un país que tiene una gran dependencia tecnológica en el extranjero, por cuestiones de seguridad, economía, etc. es necesario que ésta dependencia se disminuya. En el tema de seguridad es importante que el cifrado de datos sea llevado a cabo por IP Cores desarrollados en nuestro país, ya que éstos al ser integrados a un Sistema de Telecomunicaciones podrán contribuir a asegurar la confidencialidad de la información que se transmite, disminuyendo la injerencia de terceros.

1.1.4 Hipótesis

Es posible que con los dispositivos lógicos programables, se diseñe e implemente una arquitectura de cifrado de datos que pueda ser integrado a un Sistema de Telecomunicaciones que esté embebido en un SoC.

1.2 Objetivos

1.2.1 Objetivo General

Diseñar e implementar una arquitectura de hardware para un algoritmo de cifrado de datos, donde la información sea privada y el cual pueda ser integrado a un Sistema de Telecomunicaciones basado en un dispositivo lógico programable.

1.2.2 Objetivos Específicos

- Diseñar la arquitectura de hardware de un IP Core de cifrado de datos.
- Desarrollar en un lenguaje de descripción de hardware la arquitectura diseñada para el cifrador de datos.
- Acoplar una interfaz de comunicación de datos utilizada en Sistemas Embebidos al cifrador de datos.
- Verificar la funcionalidad del IP Core de Cifrado de datos.

- Validar la operación del IP Core de Cifrado/Descifrado de datos integrado en un dispositivo lógico programable.

1.3 Contribución

Las principales aportaciones de esta tesis son:

- La obtención de un IP Core de Cifrado AES-128/256 y un IP Core de Descifrado AES-128/256 que puedan ser integrados a un Sistema de Comunicaciones mediante la interfaz AXI4-Stream con un ancho de bus de 8-bit.
- Aumento de las capacidades de la cama de pruebas que existe en el Centro de Investigación, Innovación y Desarrollo TEcnológico (CIIDTE).
- El IP Core de Cifrado y Descifrado AES cuenta principalmente, con un 3% (de 53,200 Slices LUTs) de uso del SoC *Zynq7Z020*, una capacidad de procesamiento de 1.89 Gbps oara AES128y 2.9 Gbps para AES256 y un tiempo de ejecución de 67.4 [ns] y 87.6 [ns] respectivamente.

1.4 Esquema de la tesis

El esquema general de éste documento de tesis es el siguiente:

El capítulo 1 es una introducción que abarca el planteamiento del problema, los objetivos y la contribución.

El capítulo 2 abarca los conceptos que envuelven a un IP Core de cifrado AES en un sistema de telecomunicaciones.

El capítulo 3 describe el estado de la cuestión del cifrado sobre hardware, especificamente implementaciones en FPGAs.

El capítulo 4 describe el desarrollo de la tesis, desde el diseño del algoritmo de cifrado AES hasta IP Core AES con interfaz AXI.

El capítulo 5 se muestran los resultados en la simulación del IP Core AES y su implementación en la tarjeta de desarrollo ZedBoard.

El documento finaliza con las Conclusiones y las Referencias.

Capítulo 2

Marco Teórico

En éste capítulo se describen a detalle los elementos necesarios que constituirán un IP Core de Cifrado AES-128/256. Inicialmente, se menciona a grandes rasgos los orígenes de los Sistemas de Comunicaciones y la importancia de mantener ilegibles los mensajes que se transmiten entre ellos. Posteriormente, se menciona a la ciencia que funge como herramienta en la protección de la información transmitida, la "Criptografía", de la cual sólo se presentará aquella subrama llamada Cifrado Simétrico, que permite ocultar los datos transmitidos en un Sistema de Comunicaciones. Para ésta tesis se usó un cifrado de bloque simétrico en modo ECB, conceptos que serán descritos a detalle en las siguientes subsecciones. Además, se describe como es que el algoritmo AES funciona y que proceso se lleva sobre un mensaje, y como la finalidad de ésta tesis es la implementación hardware, es necesario revisar también aquellos rubros como los dispositivos lógicos programables y sus características que hacen posible la implementación de algoritmos como el estándar AES. Así como también se menciona como debe ser un sistema de cifrado para llamarse embebido, de modo que contenga una interfaz de comunicación conocida. Entonces, el conjunto de los conceptos aplicados dará como resultado en capítulos posteriores, un módulo de cifrado hardware AES con capacidad de integración en cualquier sistema que necesite un cifrado simétrico de bloques AES y que se capaz de comunicarse con una interfaz estándar.

2.1 Sistemas de Comunicaciones

Todos los días se está en contacto con un sistema de comunicaciones, por ejemplo cuando se habla por un teléfono móvil, cuando se escucha la radio, cuando se ve la televisión o cuando se accesa a la red más grande llamada internet. Éstos sistemas de comunicaciones están diseñados para enviar mensajes o cualquier información proveniente de un emisor para uno o más destinatarios.

Usar los sistemas de comunicaciones permite establecer una comunicación casi instantánea entre personas de distintos continentes, realizar transacciones bancarias en cualquier lugar con acceso a internet, entre otras actividades diarias desde gubernamentales, empresariales, laborales y hasta personales. El impacto de las comunicaciones en la sociedad es tan importante que es necesario mencionar su origen. La historia de las comunicaciones empleando señales electromagnéticas, se remonta al centenario 1800 donde comienzan todas las investigaciones base que dan lugar a lo que conocemos como un sistema de comunicaciones. En la Figura 2.1 se muestra una línea del tiempo con los eventos que fueron clave en el desarrollo y evolución de las comunicaciones.

Entonces, un Sistema de Comunicaciones se conforma de tres elementos: el transmisor, el canal y el receptor.

Transmisor. Convierte una señal eléctrica en una forma que pueda ser transmitida a través del medio de transmisión. En términos generales, el transmisor adapta la señal del mensaje de un modo eficiente para que viaje por el canal con un proceso llamado *modulación*[2]. Otras funciones son usadas además de la modulación, como el filtrado de la señal portadora de la información (por ejemplo, para eliminar el ruido), la amplificación de la señal modulada (que permite aumentar la amplitud de la misma), etc. que permiten realzar o transformar las características de la misma.

Canal. Es el medio físico que es usado para enviar la señal desde el transmisor al receptor, y cualquiera que sea ese medio físico, por lo que la señal se ve afectada con deformaciones, ruido e interferencias.

[1]Información obtenida de [12]

[2]Modular implica variar la señal de la información en amplitud, frecuencia o fase de una portadora. [11]

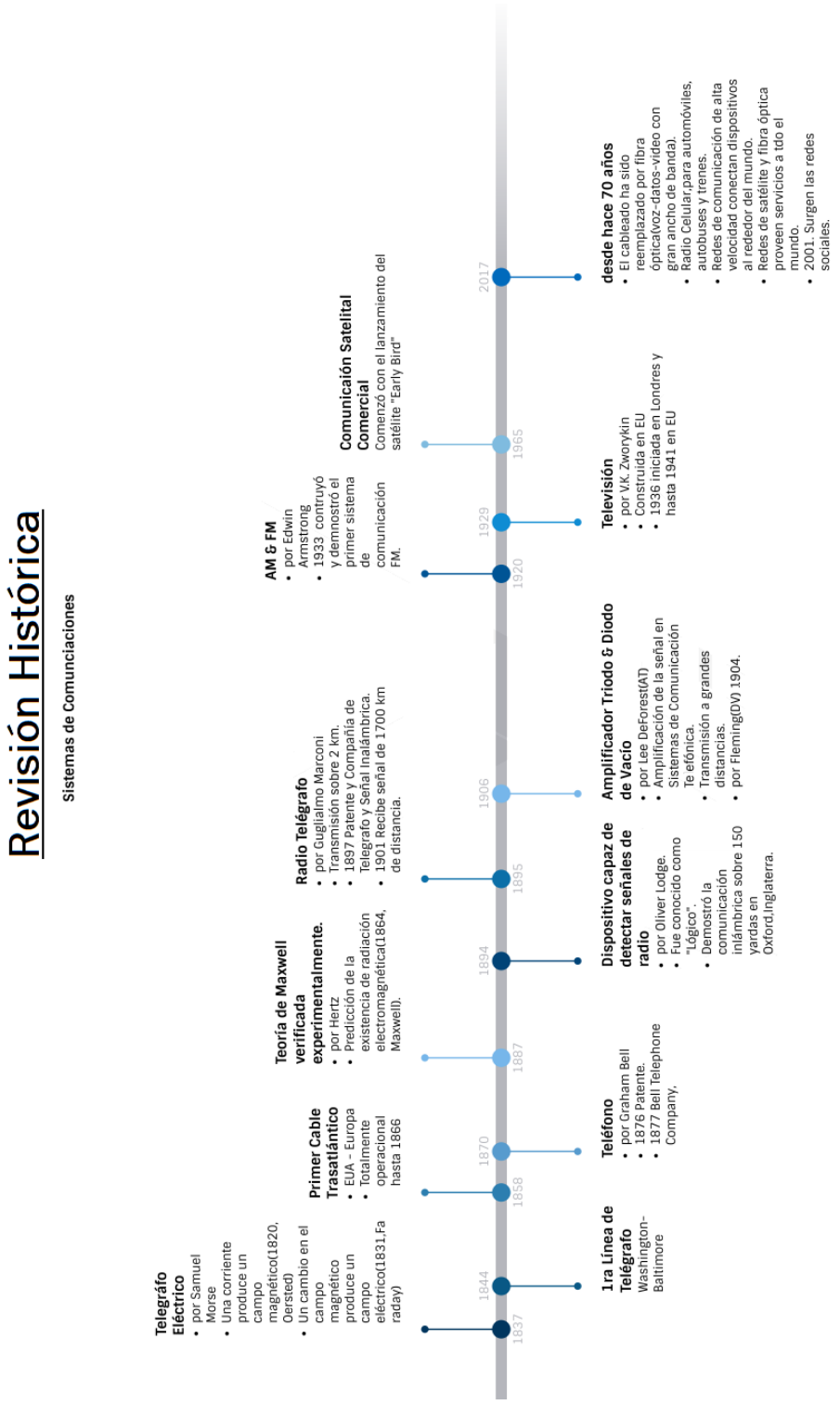


Figura 2.1: Línea de tiempo. Historia de los Sistemas de Comunicaciones[1]

Receptor. Su función es recuperar el mensaje de la señal entregada por el canal.

En un sistema de comunicaciones se puede tener como entrada señales analógicas o señales digitales dependiendo de la aplicación del sistema. Cuando una señal representa la temperatura, presión, sonido, etc. como magnitud física, se habla de una señal analógica que cambia en el tiempo y que es percibida en el medio ambiente. En muchas aplicaciones la información transmitida es digital (ejemplo: los datos en una computadora), y su transmisión cuenta con varias opciones de procesamiento de datos y otras flexibilidades que una transmisión analógica no tiene.

Las fuentes de información de las cuales se alimenta un Sistema de Comunicaciones, pueden ser de distinta naturaleza, pero es muy común que la información a transmitir sea voz, música, imágenes y datos de computadora. Si la fuente es analógica, será procesada por un convertidor analógico digital, por lo que el receptor deberá contar con un convertidor digital analógico y se pueda obtener la señal original. Si la fuente es digital, por ejemplo en el caso de los datos de computadora, es posible que la información a transmitir esté en una forma comprimida para lo cual se hará uso de un descompresor, o que se quiera transmitir la información de forma comprimida y se hagan uso de algoritmos de compresión, por ejemplo el estándar JPEG o el estándar MPEG.

Los elementos que forman un transmisor de datos procesan la fuente original para que pueda ser transmitida por un canal hacia el receptor de datos. Cuando se tiene la información digitalizada, se procesa por un *Codificador Fuente*, que se encarga de eliminar información repetitiva de la señal dando como resultado una *palabra de código fuente*. Después, dicha palabra es procesada por un *Codificador de Canal*, resultando una *palabra código de canal*. Finalmente, el modulador cambia cada símbolo de la palabra código de canal por un símbolo analógico, la secuencia de los símbolos analógicos se llama forma de onda, que será finalmente transmitida. Por su parte, el receptor cuenta con los decodificadores de los elementos del transmisor en orden inverso, el demodulador, el decodificador de canal, el decodificador fuente y el convertidor digital-análogico si fuera necesario. En la Figura 2.2 se muestra un diagrama básico de un sistema digital de comunicaciones con una breve descripción de los mismos, así como ejemplos de algoritmos que pudieran usarse en cada etapa.

Los canales por los cuales viaja la información están hechos de distintos materiales que le dan ciertas características a la transmisión. Para una trasmisión se cuenta con los siguientes medios[11]:

Cable de Par Trenzado. Un par trenzado se compone de dos conductores de cobre sólido y son susceptibles a la interferencia electromagnética, pero al trenzar los alambres se disminuyen los efectos.

Cable Coaxial. Es un alambre de cobre interior separado de un alambre de cobre estañado o de acero revestido de cobre exterior por un material aislante. Su principal característica es su mayor inmunidad al ruido, mayor ancho de banda, y tienen un mayor ancho de banda que los pares trenzados.

Fibra Óptica. Es una guía de onda dieléctrica que transmite señales luminosas que ofrece un gran ancho de banda, pérdidas casi nulas, total inmunidad a la interferencia electromagnética, peso y tamaño pequeños, resistencia y considerable flexibilidad.

Inalámbrico. Canal que soporta señales de radio y televisión, pero las señales multimedia tienen que ser moduladas sobre una frecuencia de portadora que identifica a quien transmite.

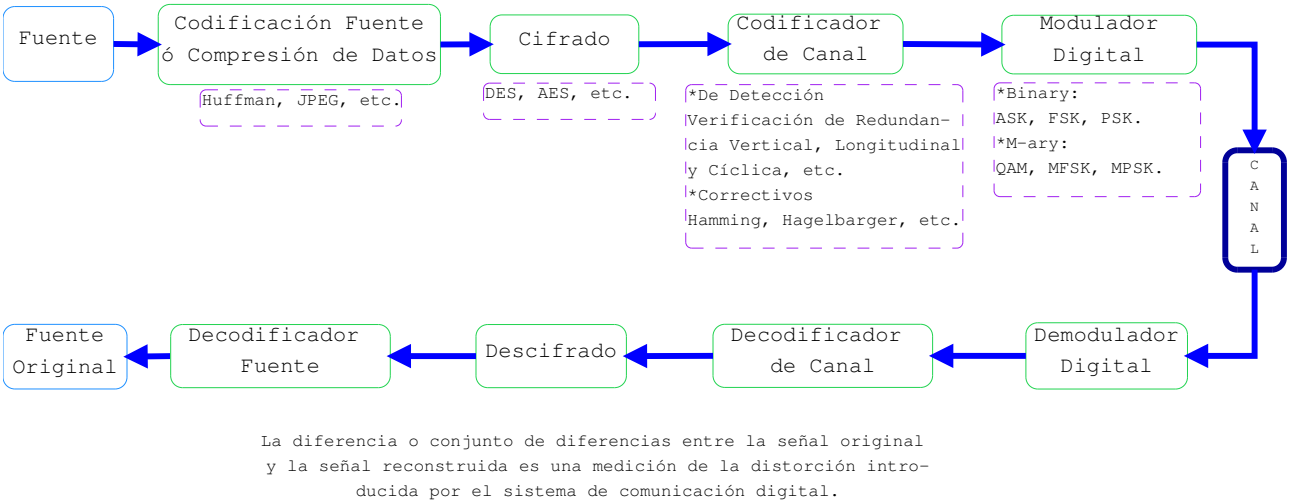


Figura 2.2: Diagrama de un Sistema de Comunicaciones[3]

La Figura 2.2 muestra una etapa llamada Cifrado, y su integración en un Sistema de Comunicaciones es opcional pues depende del tipo de información que se transmite, es decir, depende si la información es privada o pública. Por tanto, si es información delicada y a la cual sólo puede tener acceso el receptor, es necesario que se agregue una etapa de cifrado. Si la información es interceptada en el viaje a su destino, la información lucirá ilegible y en consecuencia se mantendrá la privacidad de la comunicación. Éste es un tema que se trata a detalle en la sección 2.2.

2.2 Criptografía

La seguridad en la información comprende varias rubros, por ejemplo, la seguridad en hardware, la seguridad en software, los firmwares, la información y las telecomunicaciones, éstas corrientes también son llamadas recursos de sistemas de información. Y la Seguridad Informática se encarga de la protección de los sistemas de información automatizados para lograr la integridad, disponibilidad y confidencialidad de dichos recursos. Para entender la importancia de contar con seguridad en un sistema de comunicaciones, se abarcan conceptos preliminares que describen la estructura de un sistema de información confidencial, mismos que se utilizan a lo largo de este trabajo.

La Criptología (del griego criptos: oculto y logos: tratado) es la ciencia que abarca las escrituras ilegibles, y está comprendida por las siguientes ciencias:

Criptografía. Ciencia que estudia técnicas que transforman la información de modo que no pueda entenderse a simple vista, así como protegerla de cualquier modificación y el modo de comprobar la fuente de la misma.

Criptoanálisis. Ciencia que se ocupa del análisis de un texto cifrado para obtener la información original sin conocimiento de la clave secreta, es decir, analizar la información cifrada de forma no autorizada, rompiendo así los procedimientos de cifrado establecidos por la Criptografía. El Criptoanálisis y la Criptografía son ciencias complementarias pero contrarias.

[3] Información obtenida de [11]

Esteganografía. Estudia la forma de ocultar la existencia de un mensaje, pues esconde en el interior de un mensaje otro mensaje, el cual sólo podrá ser entendido por el emisor y el receptor del mensaje, hecho que pasará inadvertido para las demás partes.

Para que un sistema pueda llamarse sistema de cifrado debe cumplir la mayoría de los siguientes principios [4], pues en la actualidad las circunstancias que rodean a la información y la tecnología que la transmite han cambiado:

(i) El mensaje cifrado, también llamado criptograma, debe ser indescifrable.

(ii) El sistema debe ser compuesto por información pública (conocida por todos), como el algoritmo que se emplea para cifrar y por información privada (sólo el emisor y el receptor la conocen). Existen varios estándares de cifrado públicos, así como esquemas de cifrado que contienen al menos una clave que servirá para cifrar/descifrar la información que sólo el emisor y/o receptor conocen.

(iii) La clave debe ser fácil de memorizar y cambiar.

(iv) El criptograma debe ser enviado por los medios de transmisión habituales.

(v) El sistema debe ser portátil y empleado por una sola persona.

(vi) El proceso de descifrado debe ser fácil de usar, su complejidad debe ser la suficiente para mantener la seguridad del sistema.

Los sistemas de cifrado hoy en día cumplen con la mayoría de los principios antes propuestos. Sin embargo, al no ser éstas características aplicables a todos los sistemas de cifrado, algunos estándares de cifrado han sido descontinuados por considerarse débiles protegiendo información. Por ejemplo, el estándar DES no es recomendado para asegurar ningún tipo de información confidencial puesto que es fácilmente descifrable por terceros, aunque su algoritmo puede enseñarse como demostración educativa en alguna escuela.

[4] Principios propuestos por Auguste Kerckhoffs en su trabajo "La Criptografía militar" en 1883

Los sistemas de cifrado contienen información privada y pública, y existe una clasificación de cifrado enfocada a las claves públicas y privadas (ver la sección 2.3). Las claves pueden ser o no fáciles de memorizar, puesto que son usadas por sistemas informáticos y no por personas. Además, entre más larga sea una clave, el sistema de comunicación será mas seguro, por ejemplo, el tiempo estimado de ruptura de la seguridad de un cifrado simétrico mediante fuerza bruta aumenta con el tamaño de la clave (ver Tabla 2.1) . Y en general, los algoritmos de cifrado son lo suficiente complejos para cifrar y descifrar todo tipo de información según sus propias características.

Tabla 2.1: Tiempo estimado para ataques de fuerza bruta en cifrado simétrico.[5]

Tamaño de la Clave	Estimación de Seguridad
56-64 bits	Un par de horas o días.
112-128 bits	Varias décadas(sin computadoras cuánticas).
256	Varias décadas aún con computadoras cuánticas.

2.3 Sistemas de Cifrado

Los sistemas de cifrado se clasifican de acuerdo al tipo de operación utilizada para obtener el criptograma, de acuerdo al número de claves para cifrar (una o dos claves) y al procesamiento del mensaje (bit a bit o en conjuntos de bits). Todos los sistemas de cifrado se basan en dos operaciones principales, una de ellas es la *sustitución* (simple sustitución de elementos del mensaje por otros) y otra es la *transposición* (reordenamiento de los elementos del mensaje), operaciones que deben ser reversibles. En cuanto al procesamiento, el mensaje es fragmentado de un modo u otro: de Flujo o de Bloque (ver secciones 2.3.2.1 y 2.3.2.2). Y la clasificación de acuerdo al número de claves es : Criptografía Simétrica o de Clave Privada y Criptografía Asimétrica o de Clave Pública (ver secciones 2.3.1 y 2.3.2).

[5]Tabla del tiempo estimado de la ruptura de la seguridad en un cifrado simétrico. [28]

Un esquema básico de un sistema de cifrado es mostrado en la Figura 2.3, donde un emisor cifra con una clave un mensaje generando un criptograma, el cual lo envía a un receptor que descifra la información con una clave. De éste modo, la información es transmitida sin ser legible.

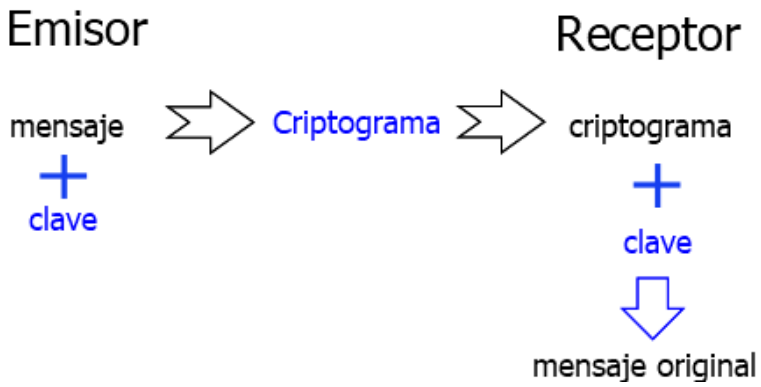


Figura 2.3: Sistema de Cifrado Básico

2.3.1 Criptografía Asimétrica

La Criptografía Asimétrica permite a las partes comunicarse de forma segura a través de una clave que es pública y otra que es privada, siempre dependientes entre sí por que están relacionadas matemáticamente. Un sistema de clave pública, descrito en 1976 por Diffie-Hellman[13], menciona que está compuesto por una clave pública y otra privada, donde la parte que las genera debe mantener en secreto la clave privada, evitando también que sea transmitida por cualquier canal, mientras que la clave pública si puede ser compartida sin restricción. En la Figura 2.4 se muestra el esquema de un sistema de cifrado asimétrico, donde el emisor cifra un mensaje x con la clave del receptor que puede ser compartida públicamente k_{pub} y lo transmite al receptor, quien lo descifra con su clave privada k_{pr} para obtener el mensaje original x.

Un ejemplo representativo del cifrado de clave pública, es la Criptografía de Curvas Elípticas, la cual es un conjunto de algoritmos de clave pública definidos en el Cryptographic Messages Syntax [14], estándar de la Internet Engineering Task Force (IEFT) que se dedica a proteger los mensajes cifrándolos, a definir la sintaxis para firmar digitalmente, a realizar funciones hash[6] y a

[6]Una función hash es una especie de huella dactilar del mensaje, identificándolo de forma unívoca.[15]

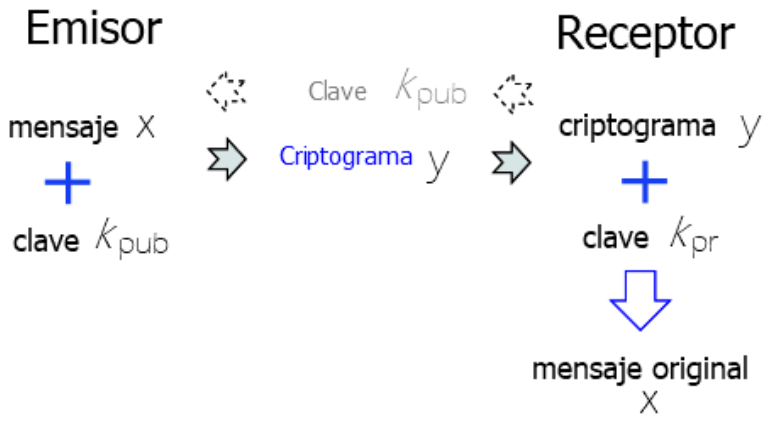


Figura 2.4: Sistema de Cifrado Asimétrico

realizar funciones MAC[7]. En otras palabras, los sistemas de cifrado de clave pública han sido desarrollados para asegurar además de la privacidad de información, su protección de copias electrónicas, de clonado, y de la destrucción de la misma, así como también, de las entradas no autorizadas al sistema de comunicación.

Un sistema basado en criptografía asimétrica, provee de establecer seguridad en las claves, puesto que una clave es pública y la clave que se necesita para descifrar el mensaje es secreta o privada. También provee de tamaños de clave mayores a los de claves de cifrado simétrico, de no repudio a la información, de identificación de las partes y del cifrado de la información, lo que lo convierte en un sistema ideal. Sin embargo, en casos prácticos el cifrado de clave pública representa mayor trabajo computacional a diferencia de los algoritmos de clave privada que cifran cien o mil veces más rápido que ellos.

2.3.2 Criptografía Simétrica

La Criptografía Simétrica usa una sola clave para cifrado y descifrado (ver Figura 2.5), pues el emisor cifra con una clave (k_{pr}) generando un criptograma, y el receptor utiliza esa misma clave para recuperar la información (x). Por lo tanto, las dos partes deben conocer la clave y siempre debe permanecer en secreto.

[7]Un checksum o MAC (Messages Authentication Code) sirve para detectar alteraciones en los mensajes, pues se almacena después de los mismos y se calcula con algoritmos ya conocidos.[15]

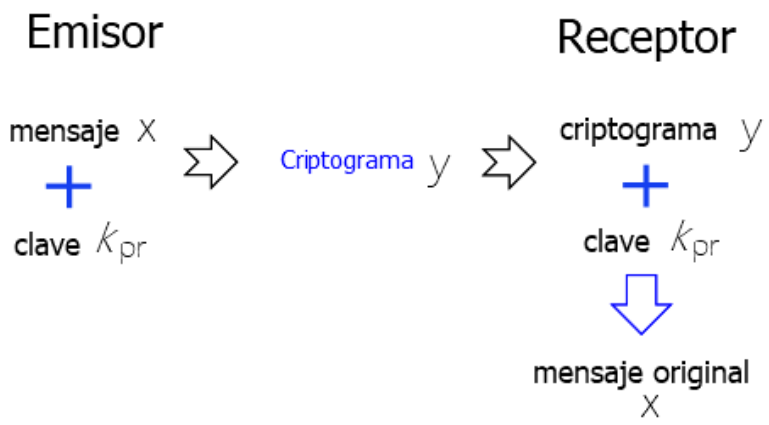


Figura 2.5: Sistema de Cifrado Simétrico

La criptografía simétrica puede aumentar su confiabilidad cuando la clave es compartida de forma privada, puesto que enviarla al receptor por el canal resulta vulnerable. También se puede verificar que la información no haya sido alterada, y se puede prevenir que se denieguen las entregas de información. Lo último sucede dependiendo la aplicación del cifrado simétrico y las necesidades a cubrir.

La Criptografía Simétrica se subdivide en Cifradores de Bloques y Cifradores de Flujo, los cifradores de bloque cifran en mensaje en bloques de tamaño fijo, y de forma similar los cifradores de flujo cifran el mensaje en bloques de tamaño 1(ver secciones 2.3.2.1 y 2.3.2.2).

Existen algoritmos de cifrado simétricos como el AES o 3DES que son seguros, rápidos y de uso general, donde el primero es el cifrador implementado en ésta tesis.

2.3.2.1 Cifradores de Flujo

Un Cifrador de Flujo opera bit a bit en un byte o cuatro bytes a la vez, con un conjunto de claves que van cambiando constantemente y que son generadas aleatoriamente por un algoritmo conocido por el emisor y receptor y pueden generar el mismo conjunto de claves. Los Cifradores de Flujo no propagan errores de transmisión, lo cual es una de sus ventajas, y una de sus desventajas es que son periódicos por naturaleza por lo que en algún momento del proceso se repetirá la clave.

Los cifradores de flujo son generalmente más rápidos que los cifradores de bloque en hardware, además de contar con una circuitería menos compleja de hardware. La unidad básica de muchos

cifradores de flujo son los LFSRs (Linear Feedback Shift Registers) y existen algunas técnicas para hacer uso de ellos. Una es usando una función combinatoria no lineal (LFSRs en paralelo), usando una función de filtrado no lineal y usando la salida de uno o más LFSRs para controlar el reloj de uno o más LFSRs. Pero en general, los LFSRs son usados en generadores de flujo de clave por varias razones, entre las cuales está su simple implementación en hardware, que produce secuencias aún con periodos largos y que su estadística tenga buenas propiedades, para que se obtenga un generador de clave seguro criptográficamente.

Hay dos tipos de cifradores de flujo:

- Cifrador síncrono de flujo. En este cifrador, el flujo de claves es generado independientemente del mensaje y del criptograma, por lo que el emisor y el receptor usan la misma clave para asegurar un descifrado correcto. Éstos cifradores no propagan errores de transmisión, pues el cambio de cada dígito no afecta el descifrado de otros dígitos. Un cifrador de bloque en modo OFB (ver subsección 2.3.3) es un ejemplo de un cifrador síncrono de flujo.
  - Cifrado de flujo aditivo binario. Cifrado en el que el flujo de clave, el mensaje y el criptograma son dígitos binarios, y la salida del cifrado y descifrado es la función XOR.
- Cifrador asíncrono de flujo. En este cifrador el flujo de clave es generado con una función y un número fijo de dígitos de cifrados anteriores, por lo que puede ser automáticamente sicronizado si los dígitos del criptograma fueran borrados o insertados. Por lo tanto, cuando algunos dígitos son borrados o insertados, es posible que el criptograma sea incorrecto despues de que se reanude el proceso de descifrado.

Otros cifradores que no son tan amigables de implementar en hardware, pero si en software son RC4 y SEAL (Sofware-optimized Encryption Algorithm) que es un cifrador de flujo aditivo binario propuesto en 1993 que representa una eficiente implementación en software para procesadores de 32-bit.

2.3.2.2 Cifradores de Bloque

Un Cifrador de Bloque divide un mensaje en bloques de tamaño fijo(usualmente de tamaño 64-bit o 128-bit), cifrando un bloque a la vez. Los cifradores de bloque se subdividen en Cifradores de Sustitución y Transposición.

Los Cifradores de Sustitución son aquellos que reemplazan símbolos o grupos de símbolos por otros símbolos o grupos de símbolos. Mientras que los Cifradores de Transposición permutan los símbolos en un bloque. Éstos cifradores por si solos no representan seguridad para los mensajes, pero si se realiza una combinación de ambos, la seguridad aumenta. Al resultado de unir una sustitución y una transposición se le llama *ronda*.

Una ronda posee las características de *confusión* y *difusión* por que contiene una sustitución y una transposición respectivamente. La *confusión* hace que la relación del mensaje con la clave tenga complejidad, y la *difusión* reordena el mensaje para evitar redundancias en el mensaje y por ende en el criptograma. Muchos Cifradores de Bloque aplican varias rondas seguidas para obtener un criptograma, además, la clave que interactúa con las rondas es una clave más larga calculada a partir de la original.

Los Cifradores de Bloque son el tipo más usado en criptografía ya que son un elemento fundamental cuando se habla de mecanismos de integridad de datos, protocolos y técnicas de autenticación y en los esquemas de firma digital. Entonces, un cifrador de bloque brinda confidencialidad en todos los sentidos, y es de vital importancia resguardar la clave con la cual se cifra un mensaje para conservar dicha característica. De igual manera, se recomienda para una seguridad perfecta, que el criptograma y los bloques de datos sean estadísticamente independientes.

2.3.2.2.1 DES-3DES

DES (Data Encryption Standard) fue de los primeros estándares en ser desarrollado comercialmente con una antigüedad de 25 años, y que abrió la pauta en el interés por la Criptografía en los investigadores de la época, así como el destape de nuevos ataques diferentes al de fuerza bruta: Criptoanálisis Diferencial, Criptoanálisis Lineal y Ataque de Davies mejorado, ataques para los

que DES se encontraba sin protección y que hicieron que finalmente se publicara el diseño del algoritmo DES.

Una de las características de DES es que su clave es de 64-bit, y debido a su corta longitud, una máquina de hardware fue capaz de demostrar que el algoritmo DES era inseguro (1998). La EFF (Electronic Frontier Foundation) construyó una máquina con la capacidad de leer mensajes cifrados con DES, además de obtener la clave con la cual fue cifrado el mensaje, ésta máquina fue construida en menos de un año y con un gasto menor a $ 250,000 dólares [3.1]. Todo lo anterior fue la causa para descontinuar el uso del cifrado DES y la necesidad de reemplazarlo.

3DES (Triple DES) es un sistema de cifrado que se compone de 3 DES secuenciales, y por cada ejecución de un DES la clave es distinta, es decir, tres claves de 64-bit. La ventaja del acoplamiento de los tres DES, fue que aumentó la fuerza en contra de los ataques que buscan la clave, sin embargo, existe una notoria disminución de la velocidad, pues éste algoritmo tarda tres veces más en obtener el cifrado de datos. Como hasta el día de hoy 3DES sigue en uso, se recomienda usar tres generadores de claves independientes.

2.3.3 Modos de Operación de los Cifradores de Bloques

Existen distintos modos de operación en los que puede trabajar un cifrador de bloques, y lo que determina cual de ellos usar es la aplicación final del cifrador. Entonces, un mensaje x formado por j-bloques de n-bit cada uno, es cifrado por una función E, y descifrado por una función inversa E^{-1}. Donde el mensaje cifrado o criptograma c se forma por el mismo número de bloques que el mensaje sin cifrar (j-bloques de n-bit cada uno).

Modo ECB (*Electronic Codebook*) Este modo de operación tiene como entrada la clave (k-bit) y el mensaje en bloques $x_1, x_2, x_3, \ldots, x_t$ de n-bit cada uno, donde t es el número de bloques del mensaje. La salida del modo es el mensaje cifrado $c_1, \ldots, c_t$, que para obtener el mensaje original se tiene que descifrar. De modo que los mismos mensajes cifrados con la misma clave, da como resultado el mismo criptograma. Y si hubera algún bit de error en un bloque cifrado, sólo se vería afectado el descifrado de ese bloque. En la Figura 2.6 se puede observar el diagrama de dicho proceso, donde $1 \leq j \leq t$, para que a partir de cada bloque del mensaje x_j se pueda obtener el bloque cifrado c_j y viceversa.

El modo ECB no es recomendado usarse en mensajes más largos que el tamaño del bloque porque los bloques son independientes entre sí. Entonces, cuando varios bloques cifrados son iguales, los bloques del mensaje son los mismos. Tampoco se recomienda usar la misma clave en más de un bloque. Y debido a que existe una correspondencia entre un mensaje sin cifrar y su criptograma, es posible que sin conocer la lógica interna del algoritmo de cifrado, que sea suceptible a diversos ataques (por ejemplo, a un ataque por sustitución).

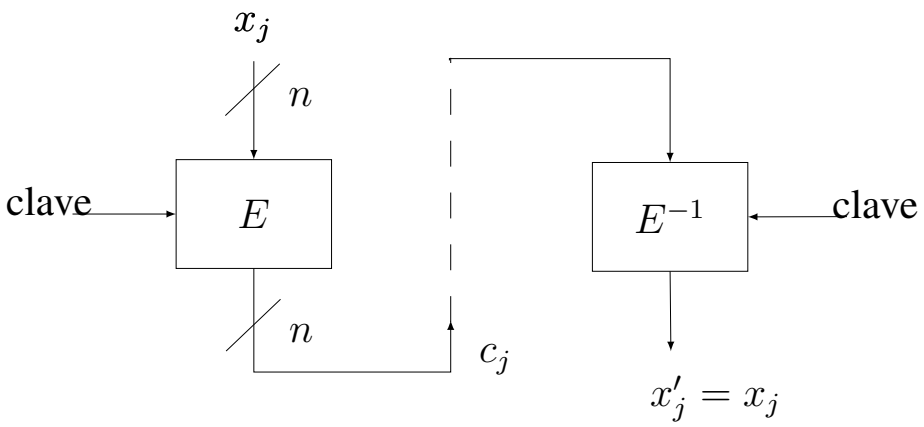


Figura 2.6: Modo ECB [8]

Modo CBC (*Cipher-Block Chaining*) Éste modo de operación tiene como entradas a la clave con k-bit, el mensaje en bloques $x_1, x_2, x_3...x_t$ de n-bit cada uno, donde t es el número de bloques del mensaje y a diferencia del modo ECB incluye una entrada más, ésta es llamada vector de inicialización IV de n-bit. La salida del modo es el mensaje cifrado $c_1, ..., c_t$ que se debe descifrar para obtener el mensaje original. En el diagrama de la Figura 2.7, donde $1 \leq j \leq t$, se muestra que a partir de la operación entre el mensaje y el último bloque cifrado $(x_j \oplus c_{j-1})$ se obtiene el bloque cifrado, y para el proceso inverso, se realiza la operación $\oplus$ entre un bloque descifrado (c_j) y el bloque descifrado anterior (c_{j-1}).

Si se tiene una misma clave, los mismos bloques del mensaje y el mismo IV, se obtendrán los mismos bloques cifrados. Pero si se cambia el IV, la clave o el primer bloque del mensaje se obtiene un criptograma diferente. En este modo se observa que los bloques cifrados dependen de los bloques cifrados anteriores y de los bloques del mensaje, por lo que si ocurre algún cambio en el orden de los bloques cifrados, el descifrado no sería posible. También, si ocurre un error en el bloque c_j pero no en c_{j+1}, c_{j+2}, el bloque x_{j+2} es descifrado sin problemas, por lo que se dice que se sincroniza automáticamente. Finalmente, si se desea

[8] Modos de operación [32]

que el cifrado cuente con integridad, es necesario que el IV sea secreto, y si no fuese el caso, sólo se contaría con confidencialidad.

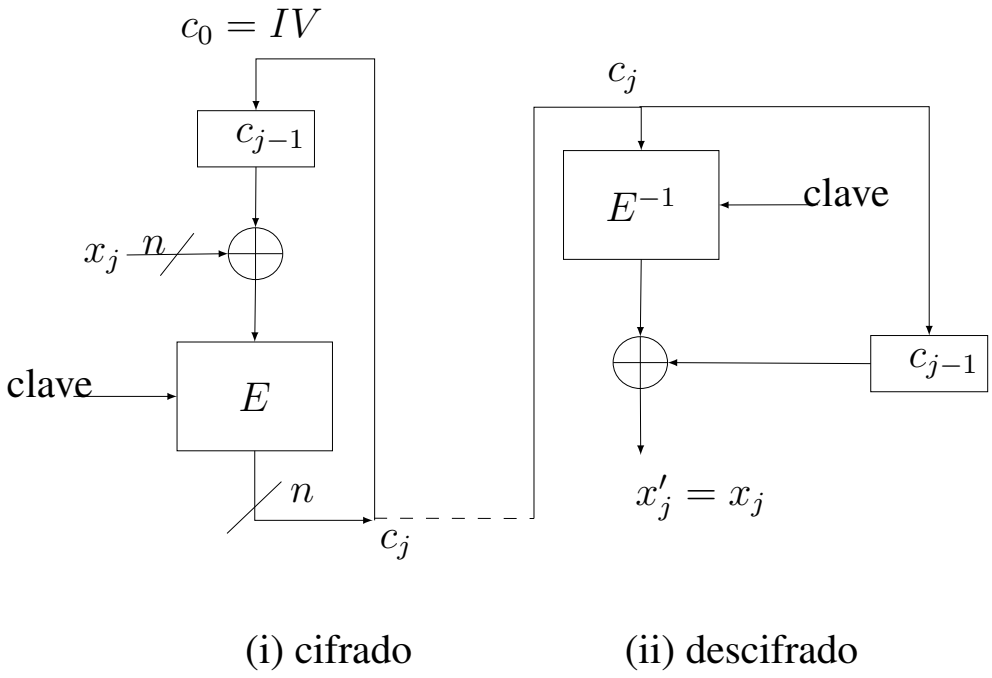


Figura 2.7: Modo CBC [9]

Modo CFB (*Cipher Feedback*) .Este modo de operación tiene como entradas una clave de k-bit, un vector de inicialización IV de n-bit y bloques del mensaje de r-bit de longitud, donde r tiene un rango de $[1 : n]$, produciendo de igual modo bloques del criptograma del mismo tamaño r-bit (ver Figura 2.8). La dependencia del bloque cifrado con el bloque del mensaje y con los bloques cifrados anteriores indica que no se puede cambiar el orden de los bloques involucrados por que se obtendría un criptograma distinto al esperado. De la misma manera que el modo CBC, cuando existe un error r-bit en un bloque cifrado c_j, se afecta el descifrado de ese bloque y de los siguientes hasta que ese bloque sea procesado por el resgistro de corrimiento y ya no se encuentre en ninguna posición. El modo CFB requiere de los $\lceil n/r \rceil$ bloques cifrados para que se sincronice automáticamente. Finalmente, el modo CFB sólo

[9] Modos de operación [32]

puede ser usado por cifradores de bloque simétricos, puesto que se usa el mismo método para cifrar que para descifrar.

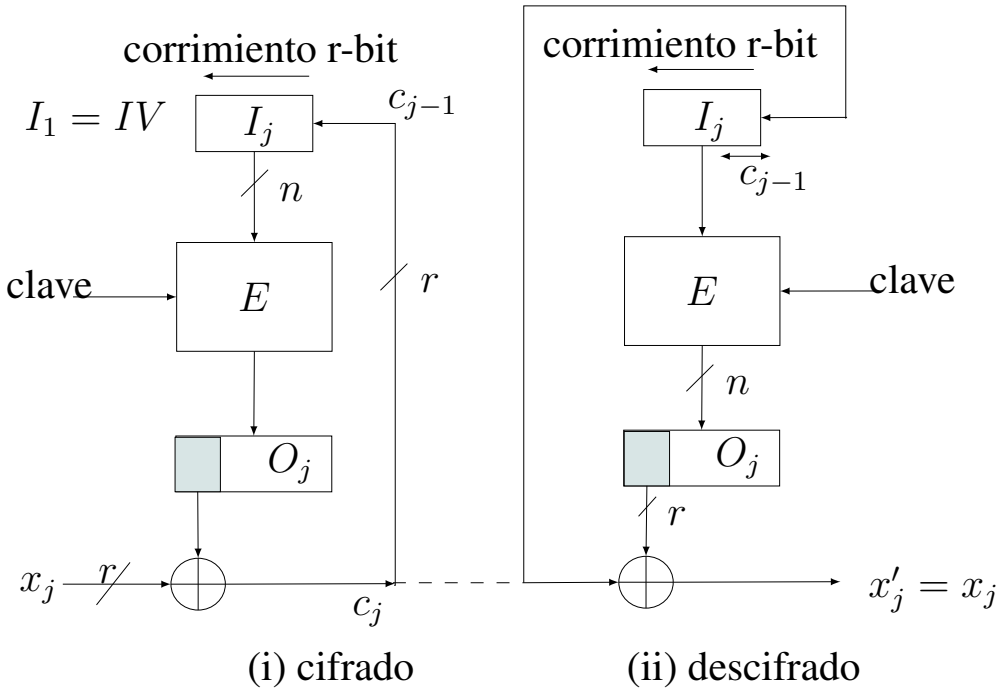


Figura 2.8: Modo CFB [10]

Modo OFB (*Output Feedback*) Este modo de operación tiene dos versiones que tienen que ver con el número de bits cifrados que son retroalimentados en el proceso de descifrado, característica que diferencia a este modo del anterior (CFB) y que opera en dos versiones: con n-bits o con $r < n$-bits. La entrada a este modo de operación es una clave de k-bit, un vector de inicialización IV de n-bit y los bloques del mensaje de tamaño r-bit, donde r tiene un rango de $[1 : n]$. La salida de este modo de operación son los bloques cifrados de tamaño r-bit $(c_1, ...c_u)$. (Ver Figura 2.9) Si existe un error en un bit o más bits de un bloque cifrado, sólo se verá afectada esa posición en el proceso de descifrado, por lo que el modo OFB se recupera pero sin la posibilidad de sincronizarse automáticamente. Se recomienda cambiar el vector de inicialización IV cuando se reuse alguna clave.

[10] Modos de operación [32]

[11] Modos de operación [32]

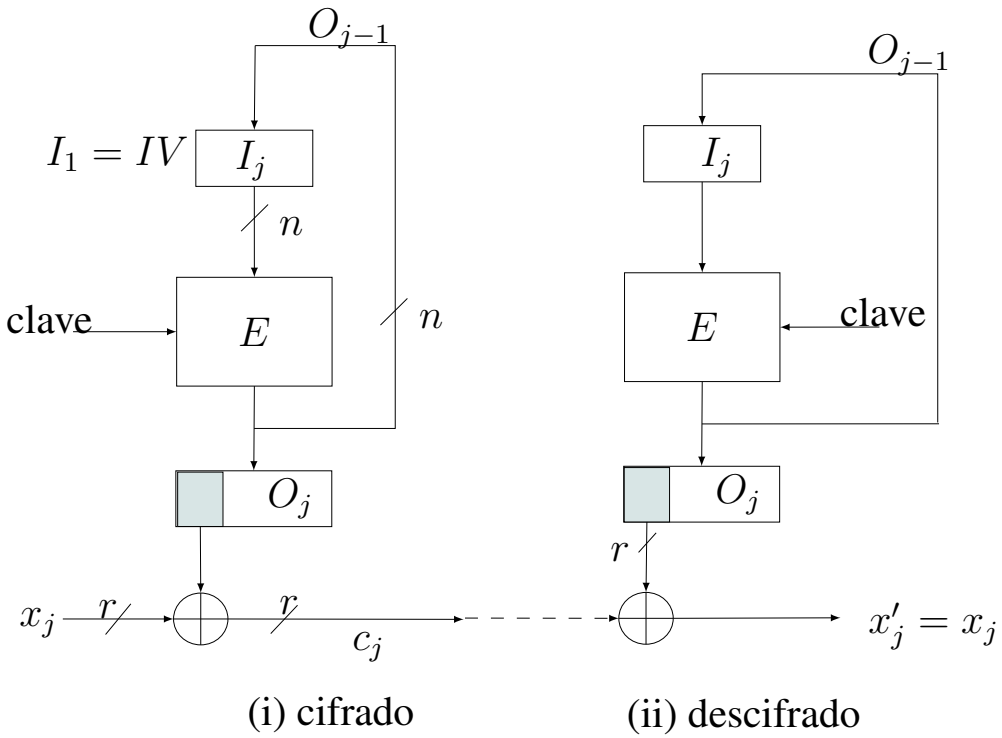
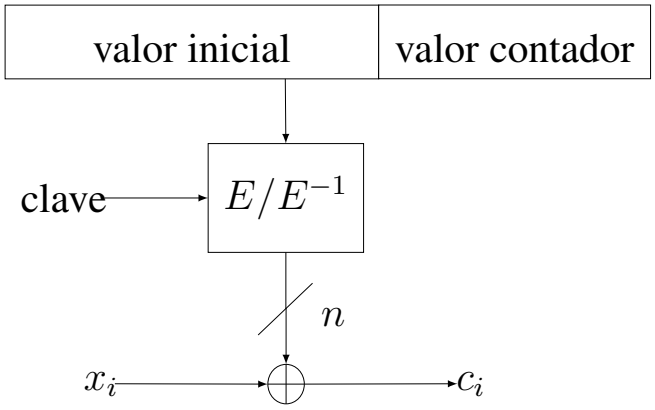


Figura 2.9: Modo OFB [1]

Modo CTR (*Counter Mode*) Este modo de operación usa un cifrador de bloque como un cifrador de flujo. La entrada al cifrador es un contador que se incrementa cada bloque de datos nuevo, por lo que es posible paralelizar el proceso siempre y cuando cada cifrado sea inicializado con el valor siguiente a su predecesor (no debe haber dos cifradores con la misma inicialización). La inicialización se compone de un vector de inicialización (IV) concatenado con un contador que es inicializado en cero, en conjunto deben sumar el tamaño de dato que reciba el cifrador de bloque. Entonces cuando se tiene un nuevo bloque de datos a cifrar, sólo se incrementa el contador, mientras que el IV permanece igual. El dato formado por la concatenación del IV con un contador inicializado en cero no tiene que ser secreto y puede enviarse junto con el primer bloque cifrado. Entonces, a diferencia del modo OFB o CFB, éste no requiere de una retroalimentación y al tener la capacidad de tener varios cifradores en paralelo, se incrementa la aceleración proporcionalmente. La aplicación de éste modo es para sistemas con altas demandas de *throughput*, como las redes en Gigabits por segundo.



(i) cifrado / descifrado

Figura 2.10: Modo CTR [12]

2.4 AES

El algoritmo Belga Rijndael[16] fue seleccionado en un concurso llevado a cabo por el NIST (National Institute for Standards and Technology) que tuvo lugar en 1997, donde se buscaba un cifrador de bloques de 128-bit de longitud para los bloques, y 128/192/256-bit de longitud para la clave. Finalmente despues de competir al lado de fuertes algoritmos como MARS, RC6, Serpent y Twofish, fue anunciado como ganador tres años despues de que se comprobara su eficiencia, seguridad y practicidad, y fue nombrado el estándar de cifrado AES (Advance Encryption Standard).

AES es un cifrador de bloques, lo que significa que se compone de un conjunto de permutaciones booleanas operando en vectores de n_b-bit, donde n puede tomar el valor de 128, 192 o 256 y b es el número de bloques. Este cifrador es especificado por un *algoritmo de cifrado* que contiene una secuencia de transformaciones que aplica al mensaje que se quiere cifrar para originar un criptograma.

El cifrador AES necesita dos datos de entrada para trabajar: un bloque de 128-bit (el mensaje o parte del mensaje) y una clave de 128/192/256-bit. La primer parte del proceso de cifrado se realiza la primera vez y cada que se utilice una clave distinta, y consiste en realizar la función

[12] Modos de operación [28]

KeyExpansion() encargada de procesar la clave de manera que a partir de ella se obtenga una subclave de mayor tamaño (**ExpandedKey**), y con la cual se cifrará el bloque del mensaje. La segunda parte consiste en una serie de rondas, y en cada ronda se procesa al mensaje (se le aplican cuatro transfomaciones matemáticas). Por ejemplo, la primera ronda incluye solo a la transformación **AddRoundKey()**, donde se realiza la operación XOR bit a bit del bloque del mensaje (matriz de 16-byte **State** correspondiente a un primer bloque de 128-bit) con el primer elemento de la subclave que debe tener el mismo tamaño y que corresponde a la clave del tamaño original.

Ahora, cuando se decide que tamaño de clave será usado para cifrar el mensaje, tres variables toman un valor fijo durante todo el cifrado de datos. Esos valores se encuentran definidos en el estándar del algoritmo [17], también mostrados en la tabla 2.4 y de los cuales dependen los cálculos dentro del algoritmo de cifrado AES.

La siguiente ronda ($round$) se repetirá $Nr-1$ veces, y en ella se incluyen las transformaciones **SubBytes()**, **ShiftRows()**, **MixColumns()** y **AddRoundKey()** (aplicadas una tras otra ese mismo orden), de las cuales se hablará con detalle en la sección 2.4.2. En la ronda final sólo se aplican las transformaciones **SubBytes()**, **ShiftRows()** y **AddRoundKey()** en ese orden, a la matriz **State** resultante de la *round* = $Nr-1$. En la Figura 2.11 se muestra el diagrama de flujo del algoritmo AES para claves de longitud 128-bit, 192-bit y 256-bit y la manera en que interactúan las rondas antes mencionadas, y que para cuando sea necesario recuperar el mensaje original se pueda invertir el proceso.

El descifrador AES necesita también de dos datos, el criptograma de 128-bit y la misma clave que se utilizó para cifrar el dato. Por lo tanto, el descifrador utiliza la misma rutina **KeyExpansion()** para expandir dicha clave. De igual forma que el cifrador, el descifrador tendrá el mismo número de rondas (Nr). En la primera ronda se encuentra la misma transformación **AddRoundKey()**, la diferencia es que comienza a operar desde el último elemento de 32-bit de la clave expandida. En las rondas intermedias operan las transformaciones **invShifRows()**, **invSubBytes()**, **AddRoundKey()** (recorre la clave expandida hasta el elemento $w[(round*Nb,(round+1)*Nb-1)$ y por último **invMixColumns()**, aplicadas en ese orden $Nr-1$ veces. En la última

[13] Diagrama de flujo basado en el pseudo-código del estándar AES [17]

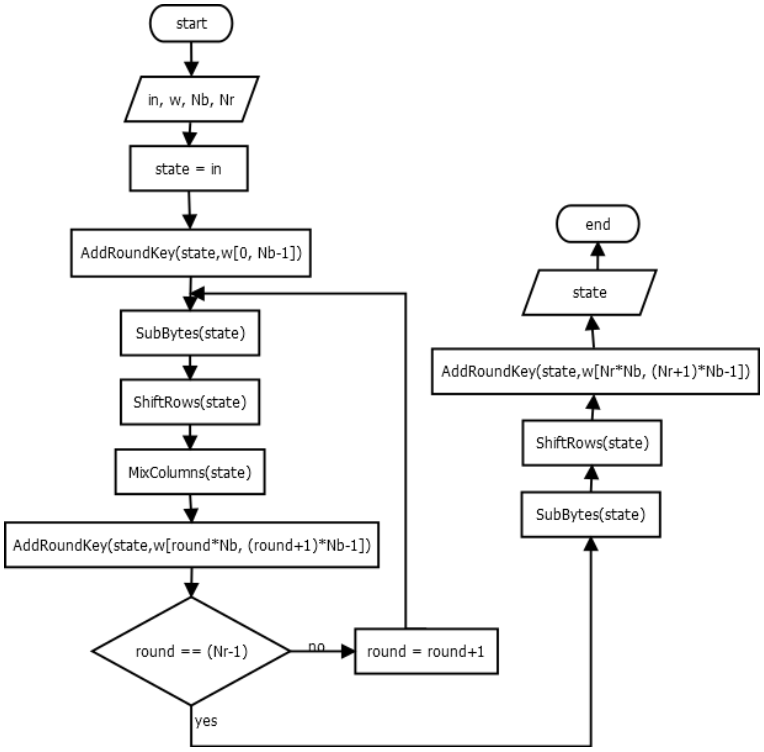


Figura 2.11: Algoritmo AES 128/192/256 de Cifrado[13]

ronda permance el orden de las transformaciones a excepción de **invMixColumns** como se ve en el diagrama de flujo de la Fig. 2.12.

[14]Diagrama de flujo basado en el pseudo-código del estándar AES [17]

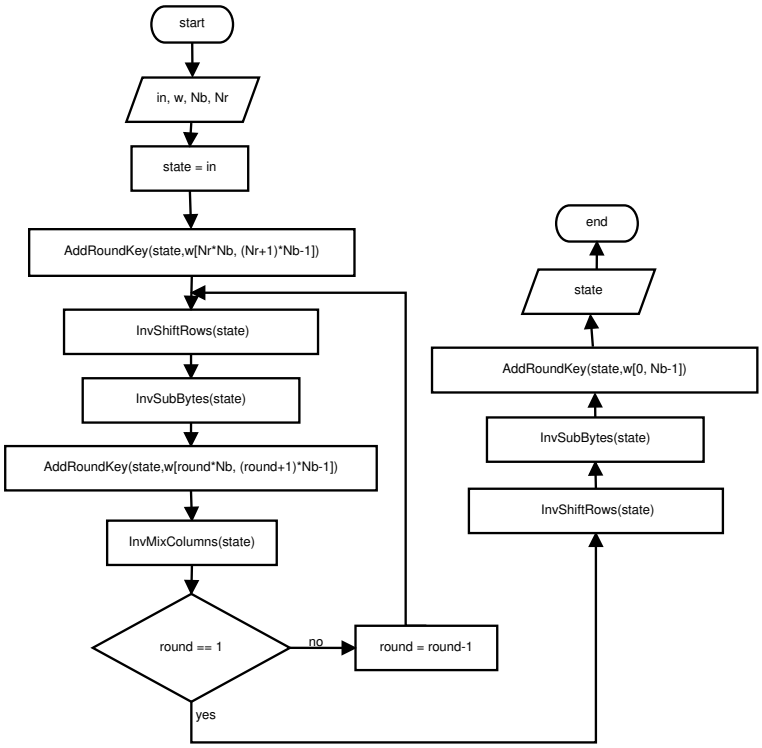


Figura 2.12: Algoritmo AES 128/192/256 de Descifrado[14]

2.4.1 Antecedentes Matemáticos

En éste momento se tienen claros los datos que requiere un sistema de cifrado AES y la interacción entre las transformaciones dentro del algoritmo, que en resumidas cuentas siempre recibirán un bloque de datos de 128-bit dividido en bytes y que será colocado como una matriz de 4×4 y por columnas, para que puedan devolver un bloque del mismo tamaño que sobreescribirá dicha matriz **State**. Entonces para explicar el funcionamiento particular de cada transformación, es necesario describir lo más claro posible la matemática que está detrás de ellas. Uno de los temas clave es la *Aritmética Modular*, por que la unidad base del algoritmo AES es el byte, y todo resultado debe ser representado sólo en 8-bit, lo que representa que se esté trabajando con un número limitado de elementos. Además, se necesita conocer que en el campo $GF(2^8)$ (ver subsección 2.4.1.2) se realizan sólo operaciones de tipo polinomial y que tienen un grado máximo de 7 en éste caso en particular y donde sus coeficientes son 1 o simplemente no existen.

Cada transformación tiene involucradas operaciones matemáticas, y donde sobresale más es en una matriz llamada S-Box, que no contiene elementos arbitrarios, si no elementos que por tener propiedades específicas, son válidos para sustituir un valor requerido (ver subsección 2.4.2.1). Por lo tanto, es muy importante saber su funcionamiento matemático y poder obtener la implementación con las características deseadas.

2.4.1.1 Aritmética Modular

La mayoría de los sistemas de cifrado, ya sea simétrico o asimétrico, están basados en aritmética dentro de un número finito de elementos, por ejemplo el conjunto de los números naturales o el conjunto de los números reales.

La **artimética modular** es entonces, la aritmética realizada en un conjunto finito de números enteros.

Definición 2.1 Operación Módulo. Sean a,r,$m \in \mathbb{Z}$, donde $\mathbb{Z}$ es el conjunto de todos los enteros y $m > 0$. Escribimos $a \equiv r \mod m$, si m divide $a - r$. Entonces m es llamado el módulo y r es llamado el resto.

Por ejemplo: $25 \equiv 1 \mod 8$, ó $42 \equiv 6 \mod 9$. Observemos que si se divide a entre m, se obtiene un cociente q y un residuo r y satisfacen que $a = mq + r$. Así que $a - r = mq$ y entonces $a \equiv r \pmod m$ según esta definición.

Ahora, si consideramos un conjunto de enteros desde cero a $m - 1$ junto con las operaciones de suma y multiplicación, tendremos una estructura algebraica llamada anillo.

Definición 2.2 Suma y multiplicación en $\mathbb{Z}_m$ (Estructura de anillo de $\mathbb{Z}_m$).
(i) Denotamos por $\mathbb{Z}_m$ al conjunto $\mathbb{Z}_m = \{0, 1, 2, ..., m - 1\}$
(ii) Definimos la *suma* de dos números en $\mathbb{Z}_m$ como el residuo de la división de la suma de los números entre el módulo. $a + b \equiv c \mod m$, $(c \in \mathbb{Z}_m)$
(iii) Definimos la *multiplicación* de dos números en $\mathbb{Z}_m$ como el residuo de la división de la multiplicación de los números entre el módulo. $a \times b \equiv d \mod m$, $(d \in \mathbb{Z}_m)$

Por ejemplo: $m = 8$ con el anillo $\mathbb{Z}_8 = \{0, 1, 2, 3, 4, 5, 6, 7\}$.

$5 + 7 = 12 \equiv 4 \mod 8$,

$5 \times 7 = 35 \equiv 3 \mod 8$

Las propiedades de los anillos son importantes puesto que en ellas se basan las características principales de un cifrado de datos, y son las siguientes[28]:

- Se dice que un anillo es cerrado por que cuando sumamos o multiplicamos dos números cualquiera, el resultado siempre está en el anillo.
- La suma y la multiplicación son *asociativas* : $a+(b+c) = (a+b)+c$, y $a \cdot (b \cdot c) = (a \cdot b) \cdot c$.
- La suma y la multiplicación son *conmutativas* : $a + b = b + a$, y $a \cdot b = b \cdot a$.
- Existe un elemento neutral 0 respecto a la suma : $a \in \mathbb{Z}_m$ mantiene que $a + 0 \equiv a \mod m$.
- Siempre existe el inverso aditivo : $a + (-a) \equiv 0 \mod m$. Por ejemplo, el inverso aditivo de $3 \mod 7$ es 4, por que $3 + 4 \equiv 0 \mod 7$. (Ver Tabla 2.2)

Tabla 2.2: Operación suma con $m = 7$

+	0	1	2	3	4	5	6
0	0	1	2	3	4	5	6
1	1	2	3	4	5	6	0
2	2	3	4	5	6	0	1
3	3	4	5	6	0	1	2
4	4	5	6	0	1	2	3
5	5	6	0	1	2	3	4
6	6	0	1	2	3	4	5

- Existe un elemento neutral 1 con respecto a la multiplicación : Para cualquier $a \in \mathbb{Z}_m$ se tiene que $a \times 1 = a$.

Tomemos $m = 2$. Entonces $\mathbb{Z}_2 = \{0, 1\}$ y notemos que $0 + 0 = 0$, $0 + 1 = 1$, $1 + 0 = 1$ y $1 + 1 = 0$. Esta operación es la misma que la conocida por XOR. Notemos también que $-1 = 1$, así que se pueden sustituir los signos negativos por positivos cuando operamos en $\mathbb{Z}_2$.

- Se dice que un número a tiene inverso multiplicativo, si existe otro (al que denotamos por a^{-1} tal que $a^{-1}a = 1$). En los enteros, $\mathbb{Z}$, 1 y -1 son sus propios inversos multiplicativos, y cualquier otro entero carece de inverso multiplicativo. Pero en $\mathbb{Z}_m$ algunos elementos a parte de 1 y -1 podrían tener inverso. Por ejemplo, en $\mathbb{Z}_{26}$, $15 \times 7 = 105 \equiv 1 \pmod{26}$, así que 7 es el inverso multiplicativo de 15. Pero si hiciéramos la tabla de multiplicación de 14 veriámos que ningún número multiplicado por 14 da por resultado 1 en $\mathbb{Z}_{26}$.

- El algoritmo extendido de Euclides calcula el máximo común divisor d de dos números a y b y coeficientes x, y, tales que $d = ax + by$. Para saber si un número a tiene inverso multiplicativo en $\mathbb{Z}_m$, y para calcularlo en caso de que exista, se aplica el algoritmo a a y m y se escribe $d = ax + my$. Si d es uno, a tiene inverso multiplicativo, y es el residuo de $x \mod m$. Si d no es uno, a no tiene inverso multiplicativo. [29, Prop. 2.1.13 y 2.1.15]

Por ejemplo, ¿Existe el multiplicativo inverso de 14 y 15 en $\mathbb{Z}_{26}$?. Desarrollando mediante el algoritmo de Euclides, el ejemplo de la Tabla 2.3-a muestra que $gcd(15, 26) = 1$ mientras que el ejemplo de la Tabla 2.3-b muestra que $gcd(14, 26) = 2$, por lo tanto sólo existe el multiplicativo inverso de 15 en $\mathbb{Z}_{26}$. Desglosando las operaciones del ejemplo de la tabla 2.3-b tenemos que al dividir m/a nos da un cociente y un residuo, a su vez, el cociente será la nueva m y el residuo la nueva a hasta que el residuo sea cero, y cuando eso pase y si la última a calculada es diferente de uno entonces no existe el multiplicativo inverso: **gcd(14,26)** $\Rightarrow$ **26** $=$ **14**$(1) + 12 \rightarrow$ **14** $= 12(1) + 2 \rightarrow 12 = 2(6) + 0$. De manera análoga se calcula el $gcd(15, 26)$ del ejemplo 2.3-a donde la última a calculada sí es uno y por tanto, existe su multplicativo inverso.

[15] donde m es el dividendo, a es el divisor, q es el cociente de dividir 26 por 15 (en el ejemplo (a)) y r es el residuo según el cálculo del mayor común divisor(gdc).

Tabla 2.3: Ejemplos que indican la existencia del multiplicativo inverso de un elemento.[15]

(a) gcd(15,26)

m	a	q	r
26	15	1	11
15	11	1	4
11	4	2	3
4	3	1	1
3	**1**	3	0

(b) gcd(14,26)

m	a	q	r
26	14	1	12
14	12	1	2
12	**2**	6	0

Continuando con el ejemplo anterior, el algoritmo extendido de Euclides dice que $gcd(15, 26) = 1$ y que $1 = 15 \cdot 7 - 4 \cdot 26$.

$$
\begin{aligned}
11 &= 26 - 15 \cdot 1 \\
4 &= 15 - 11 \cdot 1 & &= -26 + 15 \cdot 2 \\
3 &= 11 - 4 \cdot 2 & &= 26 \cdot 3 - 15 \cdot 5 \\
1 &= 4 - 3 \cdot 1 & &= 15 \cdot \mathbf{7} - 26 \cdot 4
\end{aligned}
$$

De allí se concluye que el inverso multiplicativo es 7, como se verificó antes. Sin embargo, $gcd(14, 26) = 2$, por lo que 14 no tiene inverso multiplicativo $\mod 26$.

2.4.1.2 Galois Field (2^8)

El campo finito en AES tiene 2^8 elementos que pueden ser representados por $bytes$. Éstos $bytes$ se operan bajo la aritmética en el campo $GF(2^8)$, campo que también es llamado un *Campo de Extensión* por que el orden del campo no es un número primo. Por tanto, es necesaria una notación y reglas diferentes para realizar la aritmética correspondiente, donde la aritmética polinomial es la que permite los cálculos en los *Campos de Extensión*. Entonces, para un campo de extensión $GF(2^m)$ y $m > 1$, sus elementos son representados con polinomios de grado máximo $m - 1$ y coeficientes en $GF(2)$.

Específicamente para el cifrado AES, la representación de sus elementos está dada por:

$$A(x) = a_7x^7 + \ldots + a_1x + a_0, donde\ a_i \in GF(2) = \{0, 1\},$$

y en su forma digital, puede ser representado como un $byte$:

$$A = (a_7, a_6, a_5, a_4, a_3, a_2, a_1, a_0).$$

La $suma$ y $resta$ en $GF(2^m)$ se describe como la suma y resta de coeficientes con igual potencia de x, además que sumar y restar en módulo 2 resulta en la misma operación a nivel de bit, esto es, la operación lógica XOR.

Definición 2.3 La suma y resta en un Campo de Extensión de dos elementos $A(x), B(x) \in GF(2^m)$ están dadas por:

(i) Suma

$C(x) = A(x) + B(x) = c_ix^i, \sum_{i=0}^{m-1} c_ix^i, c_i \equiv a_i + b_i \mod 2.$

(ii)Resta

$C(x) = A(x) - B(x) = c_ix^i, \sum_{i=0}^{m-1} c_ix^i, c_i \equiv a_i - b_i \equiv a_i + b_i \mod 2.$

Por su parte, la regla de multiplicación en $GF(2^8)$ está dada como sigue:

$$A(x) \cdot B(x) = (a_{m-1}x^{m-1} + \ldots + a_0) \cdot (b_{m-1}x^{m-1} + \ldots + b_0),$$

$$C'(x) = c'_{2m-2}x^{2m-2} + \ldots + c'_0,$$

$$donde\ c'_{2m-2} = a_{m-1}b_{m-1} \mod 2.$$

Definición 2.4 Sean $A(x), B(x) \in GF(2^m)$ y $P(X) \equiv \sum_{i=0}^{m} p_i x^i$, donde $p_i \in GF(2)$, polinomios que tienen como factores el número 1 y el mismo polinomio también llamados irreducibles, la multiplicación de $A(x), B(x)$ se realiza de la siguiente manera:

$C(x) = A(x) \cdot B(x) \mod P(x)$.

Un producto $C(x)$ tendrá un grado más grande que $m-1$ por lo que tiene que ser reducido, y la forma de hacerlo es dividir el producto por un polinomio irreducible y utilizar sólo el residuo. Para definir $GF(2^8)$ necesitamos hacer algo similar a la aritmética modular, pero con polinomios binarios, es decir, cuyos coeficientes están en $\mathbb{Z}_2 = 0, 1$. Consideramos el polinomio 2.1:

$$P(x) = x^8 + x^4 + x^3 + x + 1 \tag{2.1}$$

La división de polinomios binarios se realiza de foma similar a la división larga de polinomios. En este caso nos interesa calcular la división de un polinomio entre P, y calcular el residuo. Entonces, como se hizo en aritmética modular, identificaremos a un polinomio de grado *menor* que 8, es decir, no mayor que 7.

Definimos pues a $GF(2^8)$ como al conjunto de polinomios binarios de grado no mayor que 7. Por ejemplo:

$$N(x) = x^7 + x \tag{2.2}$$

$$H(x) = x^5 + x + 1 \tag{2.3}$$

Definimos la suma de elementos en $GF(2^8)$ como la suma usual de polinomios. Por ejemplo $N(x) + H(x)$,

$$N(x) + H(x) = x^7 + x^5 + 1 \tag{2.4}$$

Definimos el producto de elementos en $GF(2^8)$ como el residuo de la división entre P del producto usual de los polinomios dados. Por ejemplo $N(x) \cdot H(x) \equiv N(x)\dot{H}(x) \mod P(x)$,

$$\begin{aligned}(x^7 + x) \cdot (x^5 + x + 1) &= (x^{12} + x^8 + x^7 + x^6 + x^2 + x) \mod P(x) \\ &= x^6 + x^5 + x^4 + x^2 + x\end{aligned}$$

Finalmente notamos que un polinomio binario de grado a lo más 7 puede ser identificado con sus coeficientes, y sus coeficientes con un byte.

De manera similar a la aritmética modular, se puede ver si un polinomio en $GF(2^8)$ tiene inverso multiplicativo. El algoritmo extendido de Euclides se puede seguir usando, y la elección del polinomio P garantiza que todo polinomio no cero tiene un inverso multiplicativo. Entonces, para un elemento $A \in GF(2^m)$, su inverso A^{-1} está definido como:

$$A^{-1}(x) \cdot A(x) = 1 \mod P(x). \tag{2.5}$$

Existe una tabla (ver Tabla 2.13) que conjunta todos los inversos para $GF(2^8)$ acomodados en en filas numeradas de `0` a `f` y columnas numeradas de `0` a `f`. En ella se puede consultar el inverso de un polinomio en $GF(2^8)$ en representación hexadecimal, donde los cuatro bits más significativos del $\{byte\}_{16}$ a consultar corresponden a la fila y los cuatro bits menos significativos del $\{byte\}_{16}$ a la columna. Dicha coordenada xy ubica a un $\{byte\}_{16}$ inverso del $\{byte\}_{16}$ consultado.

	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0	00	01	8d	f6	cb	52	7b	d1	e8	4f	29	c0	b0	e1	e5	c7
1	74	b4	aa	4b	99	2b	60	5f	58	3f	fd	cc	ff	40	ee	b2
2	3a	6e	5a	f1	55	4d	a8	c9	c1	0a	98	15	30	44	a2	c2
3	2c	45	92	6c	f3	39	66	42	f2	35	20	6f	77	bb	59	19
4	1d	fe	37	67	2d	31	f5	69	a7	64	ab	13	54	25	e9	09
5	ed	5c	05	ca	4c	24	87	bf	18	3e	22	f0	51	ec	61	17
6	16	5e	af	d3	49	a6	36	43	f4	47	91	df	33	93	21	3b
7	79	b7	97	85	10	b5	ba	3c	b6	70	d0	06	a1	fa	81	82
8	83	7e	7f	80	96	73	be	56	9b	9e	95	d9	f7	02	b9	a4
9	de	6a	32	6d	d8	8a	84	72	2a	14	9f	88	f9	dc	89	9a
a	fb	7c	2e	c3	8f	b8	65	48	26	c8	12	4a	ce	e7	d2	62
b	0c	e0	1f	ef	11	75	78	71	a5	8e	76	3d	bd	bc	86	57
c	0b	28	2f	a3	da	d4	e4	0f	a9	27	53	04	1b	fc	ac	e6
d	7a	07	ae	63	c5	db	e2	ea	94	8b	c4	d5	9d	f8	90	6b
e	b1	0d	d6	eb	c6	0e	cf	ad	08	4e	d7	e3	5d	50	1e	b3
f	5b	23	38	34	68	46	03	8c	dd	9c	7d	a0	cd	1a	41	1c

Figura 2.13: Tabla de Multiplicativos Inversos [16]

Por ejemplo, el inverso de

$$D(x) = x^6 + x^3 + x^2 + x = (0100\ 1110)_2 = \{4e\}_{16} \tag{2.6}$$

se encuentra en el renglón 4, columna e:

$$D'(x) = x^7 + x^6 + x^5 + x^3 + 1 = (1110\ 1001)_2 = \{e9\}_{16}$$

[16] Multiplicativos Inversos, página 99 [28]

el cual se puede verificar de la siguiente manera:

$$(x^6 + x^3 + x^2 + x) \cdot (x^7 + x^6 + x^5 + x^3 + 1) \equiv 1 \mod P(x).$$
$$(x^6 + x^3 + x^2 + x) \cdot (x^7 + x^6 + x^5 + x^3 + 1) = x^{13} + x^{12} + x^{11} + x^{10} + x^9 + x^8 + x^6 + x^5 + x^4 + x^3 + x^2 + x$$

y haciendo la división por el polinomio irreducible $P(x)$,

$$\begin{array}{rcl}
 & & 1\ 1\ 1\ 1\ 0\ 1 \\
\hline
1\ 0\ 0\ 0\ 1\ 1\ 0\ 1\ 1 & | & 1\ 1\ 1\ 1\ 1\ 1\ 0\ 1\ 1\ 1\ 1\ 1\ 1\ 0 \\
 & & \\
 & & 1\ 0\ 0\ 0\ 1\ 1\ 0\ 1\ 1 \\
 & & \hline
 & & 0\ 1\ 1\ 1\ 0\ 0\ 0\ 0\ 0\ 1 \\
 & & \\
 & & \quad 1\ 0\ 0\ 0\ 1\ 1\ 0\ 1\ 1 \\
 & & \quad \hline
 & & \quad 0\ 1\ 1\ 0\ 1\ 1\ 0\ 1\ 0\ 1 \\
 & & \\
 & & \quad\ \ 1\ 0\ 0\ 0\ 1\ 1\ 0\ 1\ 1 \\
 & & \quad\ \ \hline
 & & \quad 0\ 0\ 1\ 0\ 1\ 0\ 1\ 1\ 1\ 0\ 1 \\
 & & \\
 & & \quad\ \ \ \ 1\ 0\ 0\ 0\ 1\ 1\ 0\ 1\ 1 \\
 & & \quad\ \ \ \ \hline
 & & \quad\ \ \ \ 0\ 0\ 1\ 0\ 0\ 0\ 1\ 1\ 0\ 1 \\
 & & \\
 & & \quad\ \ \ \ \quad 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0 \\
 & & \quad\ \ \ \ \quad \hline
 & & \quad\ \ \ \ \quad\ \ 1\ 0\ 0\ 0\ 1\ 1\ 0\ 1\ 0 \\
 & & \\
 & & \quad\ \ \ \ \quad\ \ 1\ 0\ 0\ 0\ 1\ 1\ 0\ 1\ 1 \\
 & & \quad\ \ \ \ \quad\ \ \hline
 & & \quad\ \ \ \ \quad\ \ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 1
\end{array}$$

donde el módulo es igual a 1, por lo que $D'(x)$ es efectivamente el inverso de $D(x)$:

$$D(x) \cdot D'(x) \mod x^8 + x^4 + x^3 + x + 1 = 1$$

2.4.2 Transformaciones

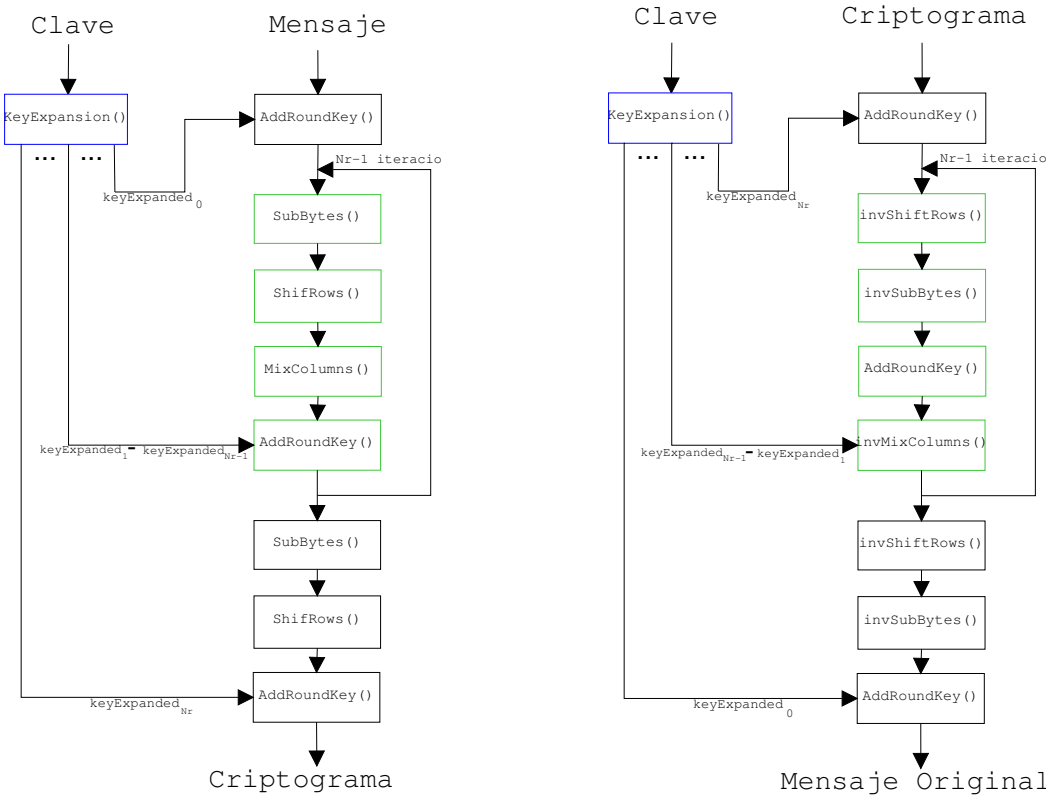
Las siguientes transformaciones matemáticas son descritas en el estándar del *National Institute of Standard for Technology* [17], las cuales en su conjunto aportan los conceptos de confusión y difusión al mensaje. Las transformaciones son aplicadas una a la vez a la matriz *State* de manera que cada una reemplaza sus valores y pueda ser usada por las siguientes transformaciones. El algoritmo de cifrado AES necesita saber cual de los tipos es, es decir, si es AES-128, AES-192 o AES-256 y así definir el valor de las constantes que usará, así como el tipo de clave que recibirá de acuerdo a su longitud en bits. Existen tres constantes llamadas $Nk, NbyNr$ definidas en la tabla 2.4, donde se muestra que su valor depende del tipo de AES de interés (AES128, AES192 y AES256).

Tabla 2.4: Combinaciones para las rondas.

AES	**Tamaño Clave (Nk-word)**	**Tamaño Bloque (Nb-word)**	**# Rondas (Nr)**
128	4	4	10
192	6	4	12
256	8	4	14

Las partes que intervienen en el cifrado AES básicamente son: una clave de cifrado, la clave de cifrado expandida, un mensaje a cifrar, las cuatro transformaciones que operan en cierto orden y un número de veces, y por último el criptograma (mensaje cifrado). La clave no es usada por el algortimo AES en su tamaño original por lo que tiene que ser expandida, y es expandida gracias a una función llamada *KeyExpansion()*, ésta función es vista a detalle en la sección 2.4.2.5. El mensaje original y el primer fragmento de la clave expandida son la entrada de la primera transformación llamada *AddRoundKey()*, y el resultado es a su vez, la entrada a una ronda 1 de $Nr - 1$ rondas. Dicha ronda conjunta a la transformación *SubBytes()*, seguida de *ShiftRows()*, seguida de *MixColumns()* y seguida por *AddRoundKey()*, que opera con un fragmento de la clave expandida.

La ronda Nr ya no incluye a la transformación *MixColumns()*, y para la transformación *AddRoundKey()* se opera con el último fragmento de la clave expandida, por lo que la salida de esta ronda es el mensaje cifrado o criptograma (ver Figura 2.14a).



(a) Cifrador AES

(b) Descifrador AES

Figura 2.14: Diagrama de Bloques AES: Cifrador-Descifrador

Por su parte, el descifrador utiliza las transformaciones inversas y que aunque la funcionalidad sea la misma, el sentido de las operaciones y los datos son diferentes, a excepción de *AddRoundKey()* que como ya se había mencionado anteriormente, la única diferencia es que la clave expandida es tomada cada 32-bit a partir del último elemento calculado, y que se muestra de manera gráfica en la Fig. 2.14b.

2.4.2.1 SubBytes / invSubBytes

La transformación *SubBytes()* es un cambio de los 16 bytes de **State** que consiste en una sustitución *byte* a *byte* por valores incluidos en una tabla llamada **S-Box**. S-Box contiene un conjunto de 256 bytes ordenados en pares coordenados (16 renglones y 16 columnas numeradas del `00` al `FF`). Entonces, cada byte de la matriz cuadrada **State** se interpreta como un par coordenado, por ejemplo el número hexadecimal `0x89` es interpretado como renglón `8` y columna `9`, el valor de tipo byte de dicha posición reemplaza el byte consultado en **State**. El orden de reemplazo es por columnas (de la columna 0 a la columna 3) de arriba hacia abajo iniciando por el elemento $S_{0,0}$. Ver Figura 2.15.

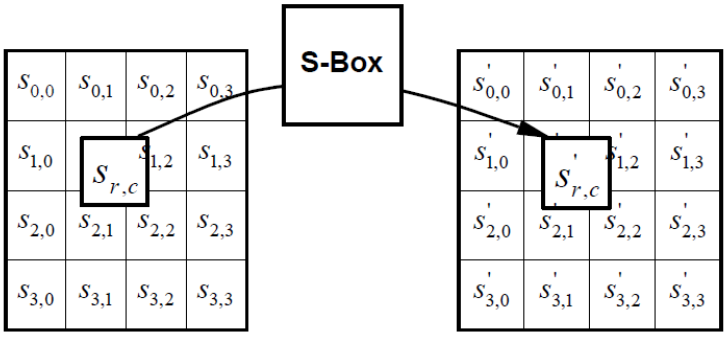


Figura 2.15: SubBytes() [17]

	00	01	02	03	04	05	06	07	08	09	0a	0b	0c	0d	0e	0f
00	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
10	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
20	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
30	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
40	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
50	53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
60	d0	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
70	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
80	cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
90	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
a0	e0	32	3a	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	79
b0	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08
c0	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
d0	70	3e	b5	66	48	3	f6	0e	61	35	57	b9	86	c1	1d	9e
e0	e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
f0	8c	a1	89	0d	bf	e6	42	68	41	99	2d	0f	b0	54	bb	16

Figura 2.16: SBOX [18]

[17] Imagen tomada del estándar NIST [17]

La tabla *S-Box* no contiene $bytes$ al azar, si no que contiene estructuras matemáticas complejas. Cuando un $\{byte\}_{16}$ es consultado, la tabla devuelve otro $\{byte\}_{16}$, y éste último es resultado de obtener el inverso multiplicativo como se vio en la sección 2.4.1.2 y posteriormente es multiplicado por una matriz y al resultado se le suma un vector de 8-bit (ver la ecuación 2.7). Por ejemplo, siguiendo el ejercicio del número $(0100\ 1110)_2 = \{4e\}_{16}$ donde su inverso multiplicativo $(1110\ 1001)_2 = \{e9\}_{16}$ fue obtenido de la tabla 2.13, sigue sustituir en la ecuación 2.7 dando como resultado $(0010\ 1111)_2 = \{2f\}_{16}$ (ver ecuación 2.8) con lo cual se verifica su existencia en la Fig. 2.18a. En consecuencia, al realizarse cada cálculo a los 256 multiplicativos inversos (ver Tabla 2.13), se obtiene la llamada *S-Box*.

$$\begin{bmatrix} s_0 \\ s_1 \\ s_2 \\ s_3 \\ s_4 \\ s_5 \\ s_6 \\ s_7 \end{bmatrix} \equiv \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} a'_0 \\ a'_1 \\ a'_2 \\ a'_3 \\ a'_4 \\ a'_5 \\ a'_6 \\ a'_7 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix} \mod 2 \tag{2.7}$$

donde a' es el inverso multiplicativo de $a \in GF(2^8)$, y s es el valor final que se coloca en *S-Box*.

[18] Imagen tomada del estándar NIST [17]

$$\begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} \equiv \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 0 \\ 1 \\ 0 \\ 1 \\ 1 \\ 1 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix} \quad \text{mod } 2 \tag{2.8}$$

Por su parte la transformación *invSubBytes()* localizada en el algoritmo de descifrado AES, también tiene como entrada a una matriz **State** a la que sustituye $byte$ a $byte$ desde una distinta **SBOX** (Fig. 2.17) para luego sobreescribir la matriz **State** con los nuevos valores.

	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0	52	09	6a	d5	30	36	a5	38	bf	40	a3	9e	81	f3	d7	fb
1	7c	e3	39	82	9b	2f	ff	87	34	8e	43	44	c4	de	e9	cb
2	54	7b	94	32	a6	c2	23	3d	ee	4c	95	0b	42	fa	c3	4e
3	08	2e	a1	66	28	d9	24	b2	76	5b	a2	49	6d	8b	d1	25
4	72	f8	f6	64	86	68	98	16	d4	a4	5c	cc	5d	65	b6	92
5	6c	70	48	50	fd	ed	b9	da	5e	15	46	57	a7	8d	9d	84
6	90	d8	ab	00	8c	bc	d3	0a	f7	e4	58	5	b8	b3	45	06
7	d0	2c	1e	8f	ca	3f	0f	02	c1	af	bd	03	01	13	8a	6b
8	3a	91	11	41	4f	67	dc	ea	97	f2	cf	ce	f0	b4	e6	73
9	96	ac	74	22	e7	ad	35	85	e2	f9	37	e8	1c	75	df	6e
a	47	f1	1a	71	1d	29	c5	89	6f	b7	62	0e	aa	18	be	1b
b	fc	56	3e	4b	c6	d2	79	20	9a	db	c0	fe	78	cd	5a	f4
c	1f	dd	a8	33	88	07	c7	31	b1	12	10	59	27	80	ec	5f
d	60	51	7f	a9	19	b5	4a	0d	2d	e5	7a	9f	93	c9	9c	ef
e	a0	e0	3b	4d	ae	2a	f5	b0	c8	eb	bb	3c	83	53	99	61
f	17	2b	04	7e	ba	77	d6	26	e1	69	14	63	55	21	0c	7d

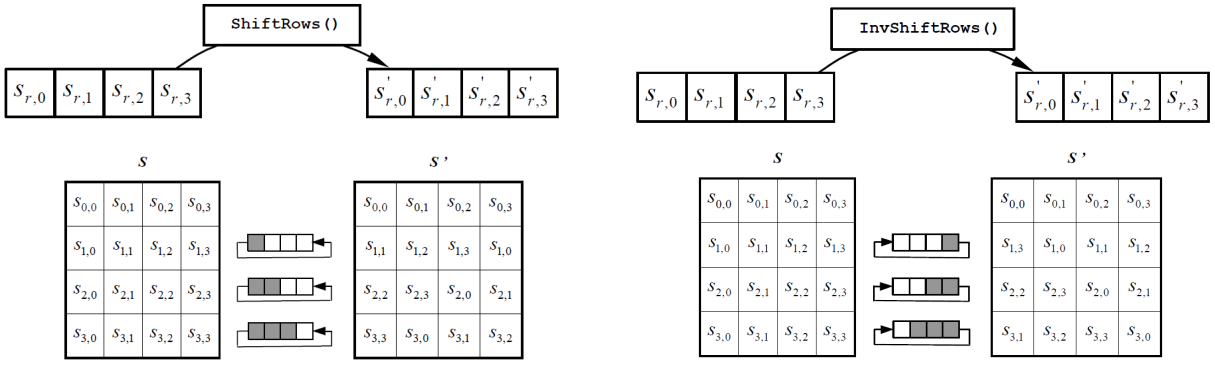
Figura 2.17: SBOX inversa.[19]

2.4.2.2 ShiftRows / invShiftRows

La transformación *ShiftRows()* es un corrimiento circular a la izquierda de los últimos tres de los renglones de la matriz **State**. Por lo que el primer renglón de **State** queda sin cambio alguno, el segundo renglón se recorre circularmente una posición a la izquierda , el tercer renglón

[19] Imagen tomada del estándar NIST [17]

se recorre circularmente dos posiciones a la izquierda y el último tenglón se recorre circularmente tres posiciones a la izquierda, tal cual lo muestra la Fig. 2.18a y se sobreescribe la matriz **State**.



(a) ShiftRows. Cifrador AES

(b) invShiftRows. Descifrador AES

Figura 2.18: ShiftRows() [20]

De igual modo, la transfomación *invShiftRows()* es una serie de corrimientos circulares de los últimos renglones de la matriz **State**, pero hacia la derecha, entonces el primer renglón queda igual, el segundo renglón se recorre una posición a la derecha, el tercer renglón se recorre dos posiciones a la derecha y el último tres posiciones a la derecha de modo que se sobreescribe la matriz **State** (ver Fig. 2.18b). En otras palabras, la transfomación *invShiftRows()* regresa la rotación que se hizo en el cifrador.

2.4.2.3 MixColumns

La transformación *MixColumns()* opera en la matriz **State** columna por columna, donde las columnas son consideradas polinomios sobre $GF(2^8)$ y son multiplicadas por $x^4 + 1$ con el polinomio $a(x) = \{03\}x^3 + \{01\}x^2 + \{01\}x + \{02\}$. *MixColumns()* es una operación que añade la mayor difusión al cifrador AES, pues cada $byte$ resultante depende de los $bytes$ originales, o los $bytes$ de entrada.

La operación puede ser escrita como una multiplicación de matrices (ver Eq. (2.9)) que desarrollada da como resultado las ecuaciones (2.10).

[20] Imagen tomada del estándar NIST [17]

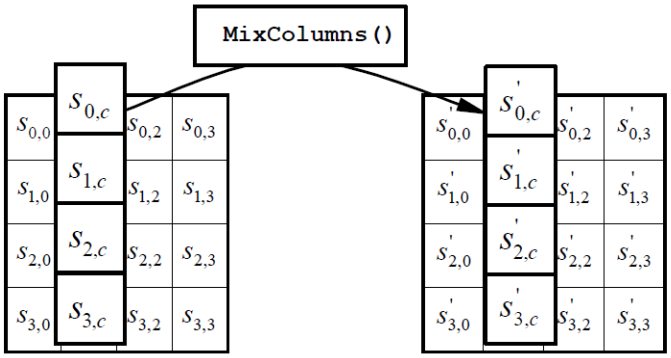


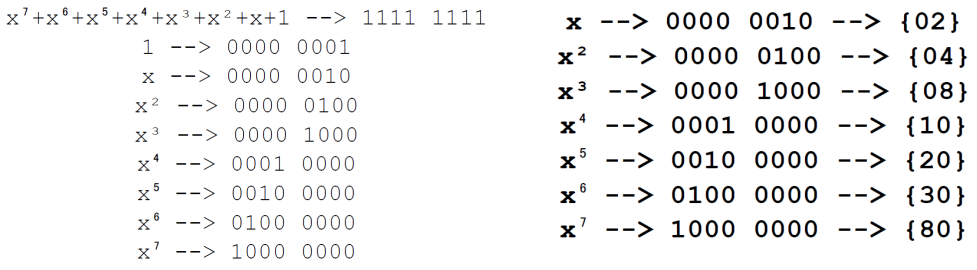
Figura 2.19: MixColumns()

$$\begin{bmatrix} S_{0,c}\prime \\ S_{1,c}\prime \\ S_{2,c}\prime \\ S_{3,c}\prime \end{bmatrix} \equiv \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} S_{0,c} \\ S_{1,c} \\ S_{2,c} \\ S_{3,c} \end{bmatrix} para\ 0 \leq c < \mathbf{Nb}. \tag{2.9}$$

$$\begin{aligned} S_{0,c}\prime &= (02 \bullet S_{0,c}) \oplus (03 \bullet S_{1,c}) \oplus S_{2,c} \oplus S_{3,c} \\ S_{1,c}\prime &= S_{0,c} \oplus (02 \bullet S_{1,c}) \oplus (03 \bullet S_{2,c}) \oplus S_{3,c} \\ S_{2,c}\prime &= S_{0,c} \oplus S_{1,c} \oplus (02 \bullet S_{2,c}) \oplus (03 \bullet S_{3,c}) \\ S_{3,c}\prime &= (03 \bullet S_{0,c}) \oplus S_{1,c} \oplus S_{2,c} \oplus (02 \bullet S_{3,c}) \end{aligned} \tag{2.10}$$

Para realizar las operaciones polinomiales se hace uso de la representación binaria, en la Fig. 2.20 vemos el tipo de polinomios con los que se opera en $Gf(2^8)$ con coeficientes igual a '1'. Entonces, si tenemos un polinomio con todos los elementos $x^7 + x^6 + x^5 + x^4 + x^3 + x^2 + x + 1$ su representación binaria-hexadecimal es `1111 1111` $-$ {`ff`} respectivamente o para representar el polinomio $0x^7 + 0x^6 + 0x^5 + x^4 + 0x^3 + 0x^2 + 0x + 0$, donde sólo tiene el elemento x^4, su representación binaria-hexadecimal es `0001 0000` $-$ {`10`} respectivamente, y siempre se utilizarán sólo 8-bit para representar un polinomio.

En la ec.(2.10) se necesita multiplicar por los valores hexadecimales de una tabla donde el polinomio `02` se multiplica por 1, por x, por x^2, por x^3, por x^4, por x^5, por x^6, por x^7 como se muestra el la Fig. 2.21a, para fines prácticos, la multiplicación por x se representa recorriendo



(a) (b)

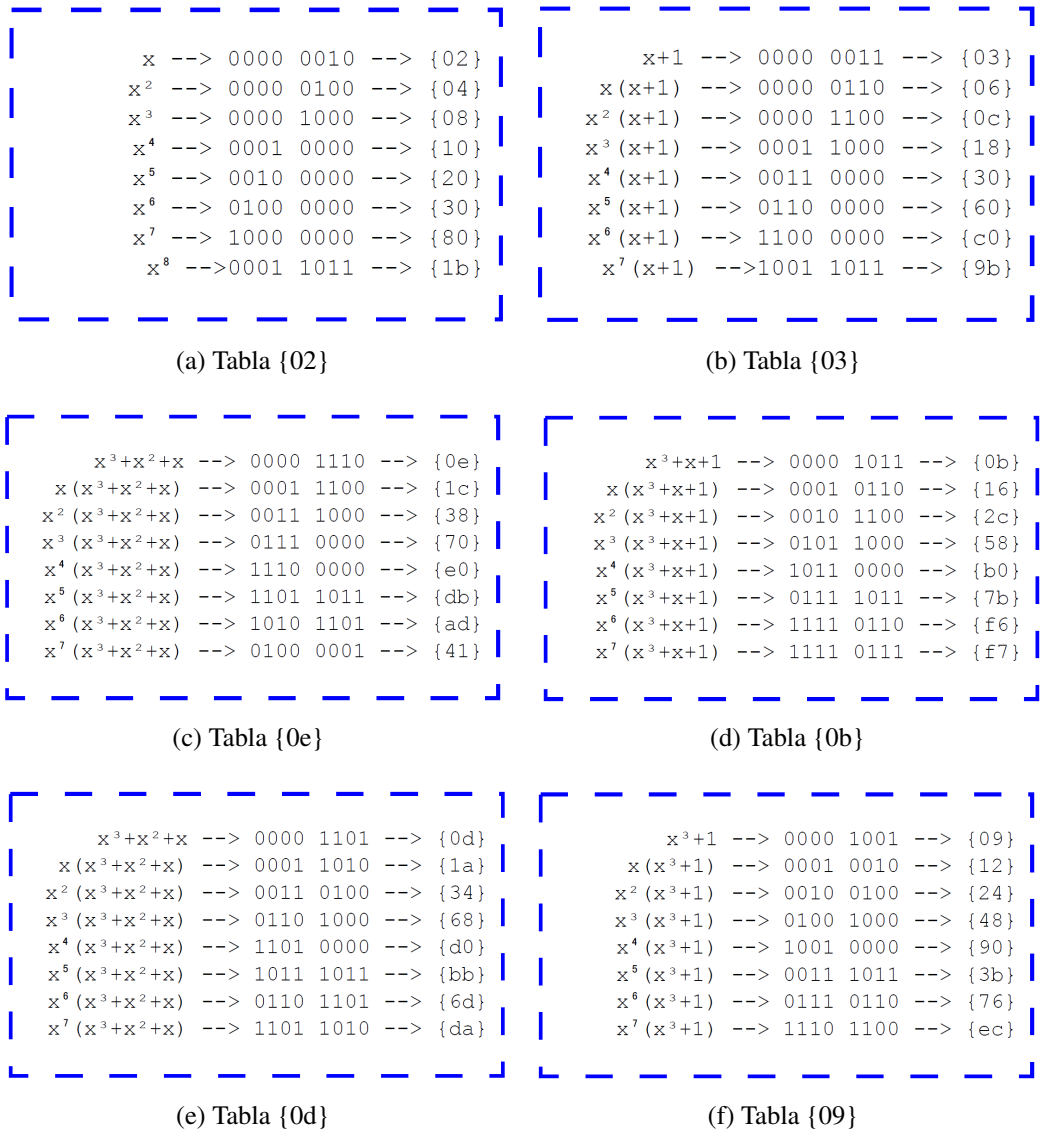
Figura 2.20: Ejemplo de la representación binaria-hexadecimal de un polinomio

los bits una posición a la izquiera, cuando un bit '1' en el MSB se recorre para obtener $x * x^7$ lo perderíamos por que quedaría en una posición que no se puede representar con 8-bit. Por lo tanto, se hace uso del polinomio irreducible ec.(2.1) visto en la sección 2.4.1.2, cuya representación hexadecimal es `1b`, por lo que se hace el corrimiento de los bits, e inmediatamente se realiza la operación $\oplus$ con `1b` lo que resulta en la multiplicación por x en $GF(2^8)$. Por ejemplo, `c0` $\bullet\ x = 1100\ 0000 \bullet x$, el primer paso es hacer el corrimiento de 1-bit a la izquierda y después aplicar la operación lógiga xor con el polinomio irreducible `1b`, entonces `c0` $\bullet\ x = 1000\ 0000 \oplus$ `1b` $= 1000\ 1011 =$ `9b` (ver Fig. 2.21b).

Ya entendida la multiplicación por x, se tiene el ejemplo de $4-byte$ {`d4`, `bf`, `5d`, `30`} que forman la columna de **State** $S_{0,1}, S_{1,1}, S_{2,1}, S_{3,1}$ respectivamente, como se muestra en la Eq. (2.11).

$$\begin{bmatrix} S_{0,c}\prime \\ S_{1,c}\prime \\ S_{2,c}\prime \\ S_{3,c}\prime \end{bmatrix} \equiv \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} d4 \\ bf \\ 5d \\ 30 \end{bmatrix} \tag{2.11}$$

$$\begin{aligned} S_{0,c}\prime &= (02 \bullet d4) \oplus (03 \bullet bf) \oplus 5d \oplus 30 \\ S_{1,c}\prime &= d4 \oplus (02 \bullet bf) \oplus (03 \bullet 5d) \oplus 30 \\ S_{2,c}\prime &= d4 \oplus bf \oplus (02 \bullet 5d) \oplus (03 \bullet 30) \\ S_{3,c}\prime &= (03 \bullet d4) \oplus bf \oplus 5d \oplus (02 \bullet 30) \end{aligned} \tag{2.12}$$



(a) Tabla {02}

(b) Tabla {03}

(c) Tabla {0e}

(d) Tabla {0b}

(e) Tabla {0d}

(f) Tabla {09}

Figura 2.21: Tablas de la multiplicación por x de un polinomio

Lo primero a realizar es la multiplicación de cada $S_{e,1}$, donde $0 \leq e < \mathbf{4}$, por la tabla de {02} y {03} según corresponda (representada por •). En la Fig. 2.22 se muestra la operación de cada polinomio por la tabla 02 y en la Fig. 2.23 por la tabla 03, donde se hace coincidir la tabla con el *byte* a multiplicar (del lsb al MSB) y se toman en cuenta para realizar la operación lógica $\oplus$ solo aquellos que coincidan con un '1' de la columna en cuestión. Con los *byte* resultantes se sustituye en la ec.(2.12) para encontrar los valores de la nueva columna $S_{e,1}\prime$.

La columna ejemplo $S_{e,1}$ que opera la transformación *Mixcolumns()* es sustituida por la columna generada $S_{e,1}\prime$ (ver ec. (2.13)), éste proceso se repite para las tres columnas restantes de la matriz **State**.

$$
\begin{aligned}
S_{0,c}\prime &= \texttt{b3} \oplus \texttt{da} \oplus \texttt{5d} \oplus \texttt{30} = \texttt{04} \\
S_{1,c}\prime &= \texttt{d4} \oplus \texttt{65} \oplus \texttt{e7} \oplus \texttt{30} = \texttt{66} \\
S_{2,c}\prime &= \texttt{d4} \oplus \texttt{bf} \oplus \texttt{ba} \oplus \texttt{50} = \texttt{81} \\
S_{3,c}\prime &= \texttt{67} \oplus \texttt{bf} \oplus \texttt{5d} \oplus \texttt{60} = \texttt{e5}
\end{aligned}
\tag{2.13}
$$

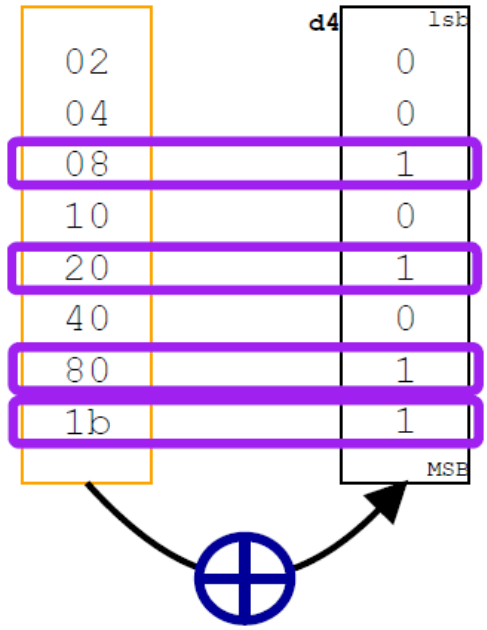
El transformación *invMixColumns()* que se centra en el descifrador corresponde a la transformación inversa de *MixColumns()* . Las operaciones realizadas son también sobre el campo $GF(2^8)$ donde la diferencia son las ecuaciones pues los polinomios con los cuales se multiplica ($\bullet$) son cuatro y todos distintos (ver ec.(2.15)).

$$
\begin{bmatrix} S_{0,c}\prime \\ S_{1,c}\prime \\ S_{2,c}\prime \\ S_{3,c}\prime \end{bmatrix} \equiv \begin{bmatrix} 0e & 0b & 0d & 09 \\ 09 & 0e & 0b & 0d \\ 0d & 09 & 0e & 0b \\ 1b & 0d & 09 & 0e \end{bmatrix} \begin{bmatrix} S_{0,c} \\ S_{1,c} \\ S_{2,c} \\ S_{3,c} \end{bmatrix} \; para\; 0 \leq c < \mathbf{Nb}. \tag{2.14}
$$

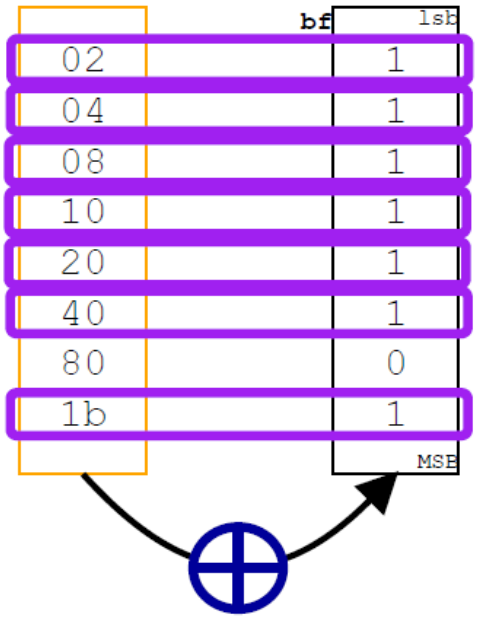
$$
\begin{aligned}
S_{0,c}\prime &= (\texttt{0e} \bullet S_{0,c}) \oplus (\texttt{0b} \bullet S_{1,c}) \oplus (\texttt{0d} \bullet S_{2,c}) \oplus (\texttt{09} \bullet S_{3,c}) \\
S_{1,c}\prime &= (\texttt{09} \bullet S_{0,c}) \oplus (\texttt{0e} \bullet S_{1,c}) \oplus (\texttt{0b} \bullet S_{2,c}) \oplus (\texttt{0d} \bullet S_{3,c}) \\
S_{2,c}\prime &= (\texttt{0d} \bullet S_{0,c}) \oplus (\texttt{09} \bullet S_{1,c}) \oplus (\texttt{0e} \bullet S_{2,c}) \oplus (\texttt{0b} \bullet S_{3,c}) \\
S_{3,c}\prime &= (\texttt{0b} \bullet S_{0,c}) \oplus (\texttt{0d} \bullet S_{1,c}) \oplus (\texttt{09} \bullet S_{2,c}) \oplus (\texttt{0e} \bullet S_{3,c})
\end{aligned}
\tag{2.15}
$$

[21] Columna ejemplo tomada del documento oficial del estándar AES [17]

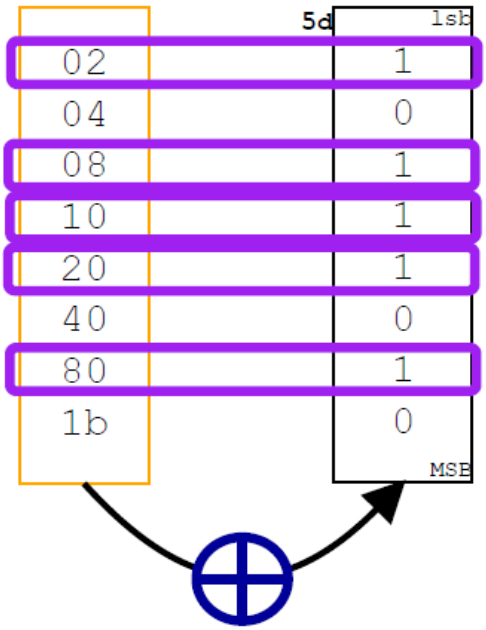
[22] Columna ejemplo tomada del documento oficial del estándar AES [17]



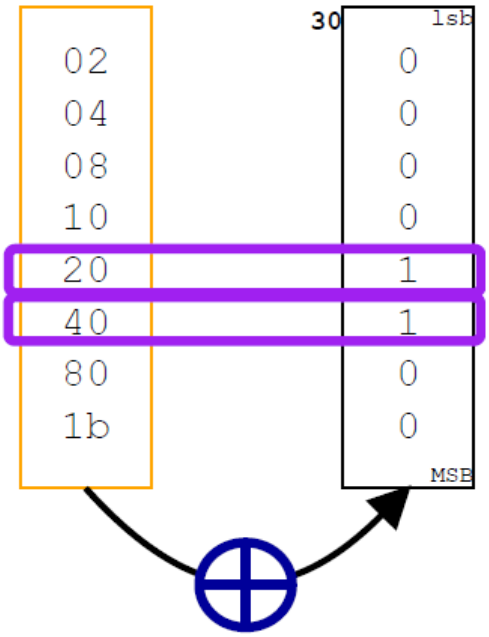
(a) $02 \bullet d4 = 08 \oplus 20 \oplus 80 \oplus 1b = b3$



(b) $02 \bullet bf = 02 \oplus 04 \oplus 08 \oplus 10 \oplus 20 \oplus 40 \oplus 1b = 65$

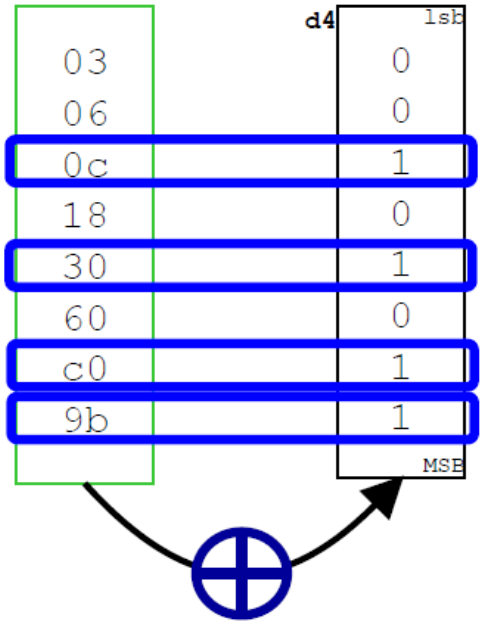


(c) $02 \bullet 5d = 02 \oplus 08 \oplus 10 \oplus 20 \oplus 80 = ba$

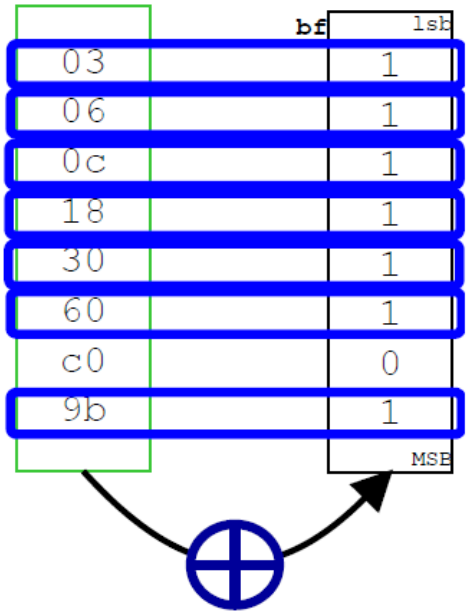


(d) $02 \bullet 30 = 20 \oplus 40 = 60$

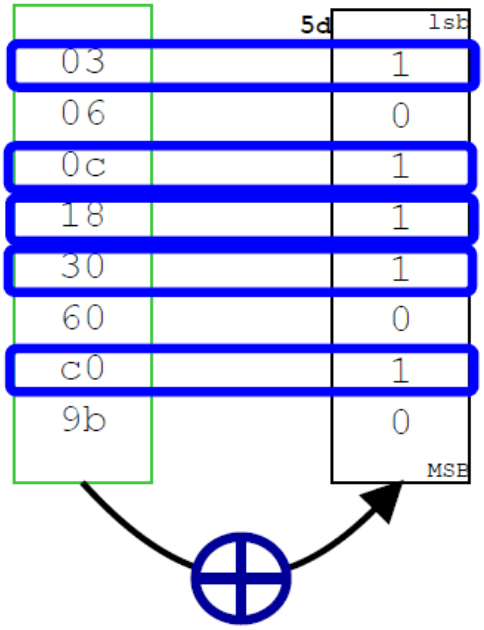
Figura 2.22: Multiplicación de los elementos de la primera columna ejemplo de **State** $S_{1,c}$ con $\{02\}$.[21]



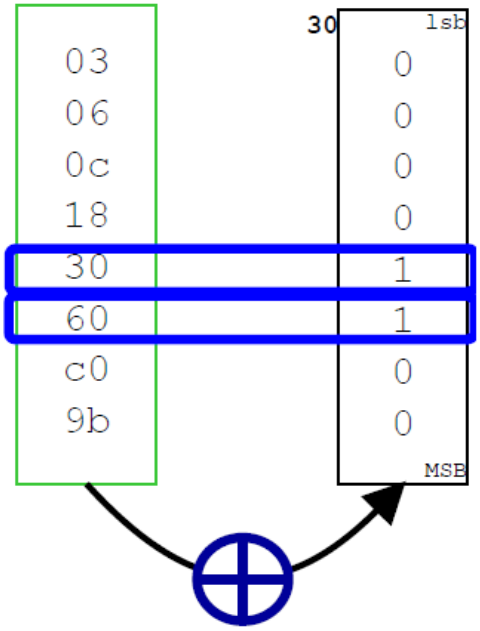
(a) $03 \bullet d4 = 0c \oplus 30 \oplus c0 \oplus 9b = 67$



(b) $03 \bullet bf = 03 \oplus 06 \oplus 0c \oplus 18 \oplus 30 \oplus 60 \oplus 9b = da$



(c) $03 \bullet 5d = 03 \oplus 0c \oplus 18 \oplus 30 \oplus c0 = e7$



(d) $03 \bullet 30 = 30 \oplus 60 = 50$

Figura 2.23: Multiplicación de los elementos de la primera columna ejemplo de **State** $S_{1,c}$ con $\{32\}$.[22]

2.4.2.4 AddRoundKey

La transformación *AddRoundKey()* opera en la matriz **State** aplicando la operación lógica *XOR* bit a bit con cada ronda de clave. Es decir, a cada columna de longitud 32-bit se le aplica la operación *XOR* con un segmento de 32-bit de la clave expandida. El segmento de la clave a seleccionar, o también llamado ronda de clave, depende en que posición del algoritmo AES se esté operando. Si se está en la ronda inicial, a las cuatro columnas de *State* de 32-bit cada una, se les aplica la operación *XOR* con las primeras 4-word de la **ExpandedKey**, por lo que las rondas subsecuentes irán operando con los 128-bit siguientes de **ExpandedKey** y así sucesivamente. Ver Figura 2.24

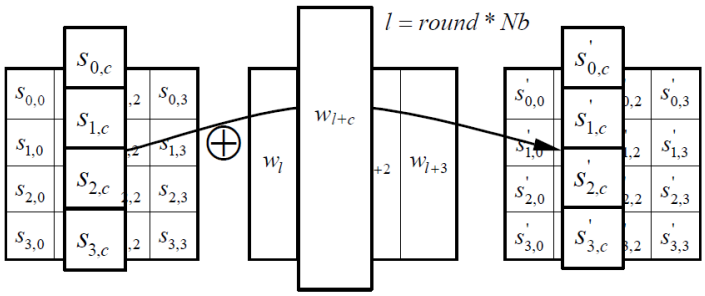


Figura 2.24: AddRoundKey [23]

[23] Imagen tomada del estándar NIST [17]

2.4.2.5 KeyExpansion

La rutina *KeyExpansion()* resulta en una versión expandida de la clave original (ver el algoritmo en la Figura 2.25). La expansión de la clave genera un total de **Nb** $\cdot$ (**Nr** $+ 1$) conjuntos de 32-bit (1-*word*), según el valor que tome Nb, Nr y Nk respecto al tamaño de la clave (ver Tabla 2.4). Si se piensa en un arreglo w_i, donde $0 <= i < Nb \cdot (Nr + 1)$, los elementos w_0 hasta w_{Nk-1} corresponden a la clave dada, así que para calcular los siguientes (128/192/256)-bit, se sigue con el caso $i = Nk$ donde w_{i-1} y de w_{i-Nk} son necesarios. Por ejemplo, para calcular el elemento w_4 (para $i \geq Nk = 4$) se necesita el elemento anterior w_3y el elemento w_0, procedimiento que se repite hasta el elemento $w_{[Nb\cdot(Nr+1)]-1}$.

```
KeyExpansion(byte key[4*Nk], word w[Nb*(Nr+1)], Nk)
begin
   word  temp

   i = 0

   while (i < Nk)
      w[i] = word(key[4*i], key[4*i+1], key[4*i+2], key[4*i+3])
      i = i+1
   end while

   i = Nk

   while (i < Nb * (Nr+1)]
      temp = w[i-1]
      if (i mod Nk = 0)
         temp = SubWord(RotWord(temp)) xor Rcon[i/Nk]
      else if (Nk > 6 and i mod Nk = 4)
         temp = SubWord(temp)
      end if
      w[i] = w[i-Nk] xor temp
      i = i + 1
   end while
end
```

Figura 2.25: Pseudo-código KeyExpansion [24]

En el algoritmo de *KeyExpansion()* de la Figura 2.26 operan dos transformaciones que se aplican cuando ya se tienen los primeros Nk elementos de w. *RotWord()* realiza una rotación a la izquierda de la palabra de entrada compuesta de 4-byte, por ejemplo, para una palabra $\{p_0, p_1, p_2, p_3\}$ el resultado de la rotación sería $\{p_1, p_2, p_3, p_0\}$. *SubWord()* tiene la misma funcionalidad que *SubBytes()* 2.4.1.1 en tanto que se busca en la tabla *SBox* por pares coordenados, la diferencia radica

[24] Imagen tomada del estándar NIST [17]

en que en vez de buscar $byte$ por $byte$, se buscan cuatro $bytes$ de una sola vez, por lo que será reemplazada una palabra completa (entrada: $\{p_0, p_1, p_2, p_3\}$, salida: $\{p'_0, p'_1, p'_2, p'_3\}$).

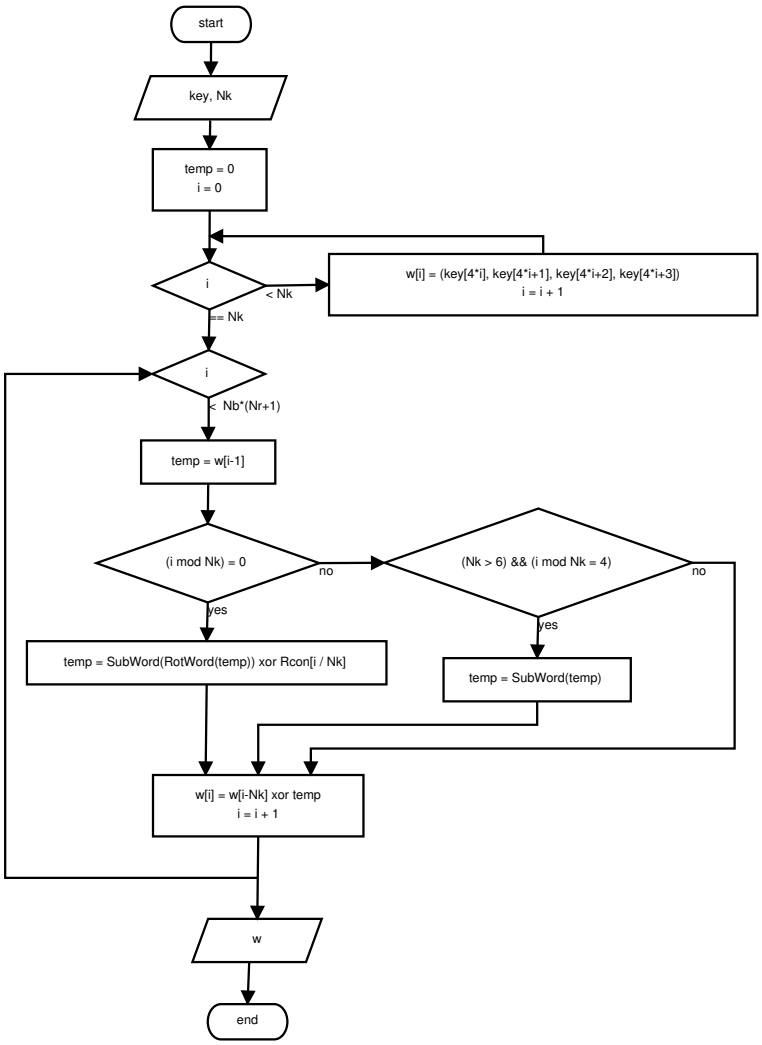


Figura 2.26: Diagrama de Flujo de la rutina de clave KeyExpansion()

Dentro del algoritmo también se encuentra $Rcon$, que es un arreglo constante en donde cada uno de sus elementos es un conjunto de cuatro numeros hexadecimales (2-byte cada uno) acomodados de la forma $[x^{j-1}, \{00\}, \{00\}, \{00\}]$, donde $j = i/Nk$ y el índice i corresponde a cierto elemento de w a partir de $i = Nk$ hasta $i < Nb \cdot (Nr + 1)$, pues los primeros Nk valores ya son conocidos. El índice j se encuentra en el rango $j = 1$ hasta $j = 16$ pero el cifrado AES opera hasta $j = 11$. Las potencias de x que se listan en el byte más significativo están siempre en $GF(2^8)$, por lo que el arreglo $Rcon[j - 1]$ se calcula como menciona el siguiente párrafo.

El byte más significativo de cada elemento de Rcon se compone de potencias de x^{j-1}, potencias que se calculan con la multiplicación polinomial en $GF(2_8)$. Rcon[0] equivale a $x^0 = 1$, la siguiente potencia se calcula multiplicando por x que equivale a $x(1) = x$, entonces la que sigue se calcula multiplicando por x que equivale a $x(x) = x^2$, y cada vez se irá multiplicando por x para obtener las potencias necesarias, siempre y cuando su representación binaria pueda ser contenida en 8-bit. Cuando la multiplicación por x sobre pasa los 8-bit, de manera equivalente se obtiene el módulo $x^8+x^4+x^3+x+1$ (`0x11b`) de dicha representación binaria. Por ejemplo, la representación binaria de 9-bit de x^8 es 1 0000 0000, entonces se procede a obtener $x^8 \mod (x^8+x^4+x^3+x+1)$ como sigue:

$$
\begin{array}{rcccccccc|cccccccccc}
 & & & & & & & & & & & & & & & & & 1 \\
\cline{10-19}
1 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
 & & & & & & & & & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 \\
\cline{10-19}
 & & & & & & & & & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1
\end{array}
$$

donde el residuo binario $11011 (x^4+x^3+x+1$ ó `0x1b`) corresponde al resultado de la operación módulo. Por lo tanto, $x^4 + x^3 + x + 1 \equiv x^8 \mod (x^8 + x^4 + x^3 + x + 1)$, y de igual forma con los elementos restantes del arreglo (ver 2.5).

El arreglo de la clave expandida w tiene ahora los primeros Nk elementos, para calcular el elemento siguiente, se guarda el elemento anterior w_{i-1} es guardado en una variable temporal $temp$ para después verificar si $i \mod Nk = 0$, si es así, a $temp$ se le aplica la transformación $RotWord()$ y a la palabra rotada se le aplica $SubWord()$ para finalmente hacer la operación XOR con el correspondiente $Rcon[i/Nk]$. Si el módulo anterior es diferente a cero, pero igual a cuatro y $Nk > 6$, caso que aplica sólo para AES-256, sólo se le aplica $SubWord()$ a $temp$. Por último, se realiza operación binaria XOR entre el elemento $w[i - Nk]$ y $temp$ para obtener el elemento $w[i]$ y se incrementa en 1 i hasta $(i < Nb * (Nr + 1))$. Cuando la clave ha sido expandida, será usada por el cifrador AES en intervalos de 128-bit, según el número de rondas Nr que realice.

Entonces, se muestra un breve ejemplo de la expansión de las primeras dos subclaves de una clave de 128-bit en la Figura 2.27. Siguiendo el algoritmo, la clave de 128-bit se compone de

Tabla 2.5: Arreglo Rcon, de los cuales sólo los diez primeros elementos son usados por AES.

j	**Rcon[j-1]**	**MSB** x^{j-1}	**MSB Binario**	**MSB Hexadecimal**	**Rcon**
1	Rcon[0]	$x^0 = 1$	00000001	{01}	{01000000}
2	Rcon[1]	$x^1 = x$	00000010	{02}	{02000000}
3	Rcon[2]	x^2	00000100	{04}	{04000000}
4	Rcon[3]	x^3	00001000	{08}	{08000000}
5	Rcon[4]	x^4	00010000	{10}	{10000000}
6	Rcon[5]	x^5	00100000	{20}	{20000000}
7	Rcon[6]	x^6	01000000	{40}	{40000000}
8	Rcon[7]	x^7	10000000	{08}	{80000000}
9	Rcon[8]	x^8	00011011	{1b}	{1b000000}
10	Rcon[9]	x^9	00110110	{36}	{36000000}
11	Rcon[10]	x^{10}	01101100	{6c}	{6c000000}
12	Rcon[11]	x^{11}	11011000	{d8}	{d8000000}
⋮	⋮	⋮	⋮	⋮	⋮

4-word por lo que cada *word* corresponde a los primeros 4 elementos o subclaves de la clave expandida, colocándose de la siguiente manera: los 32-bit MSB de la clave son el elemento 0 de la clave expandida, los siguientes 32-bit son el elemento 1, y así sucesivamente hasta llegar al elemento 3. Para calcular el elemento siguiente, en éste caso el 4, se guarda el último elemento (3) en una variable temporal y a la cual posteriormente se le aplica la transformación $RotWord()$ y a ésta la transformación $SubWord()$ para posteriormente realizar la operación XOR con el primer elemento del arreglo **Rcon** (`0x01000000`), donde el valor resultante sustituye al de la variable temporal anterior. Finalmente el resultado de aplicar XOR entre la variable temporal y el elemento 0 ($w[i-Nk] \longrightarrow w[4-4]$) es igual a la subclave 4 ($w[4]$). Éste procedimiento se va a repetir cada cuatro iteraciones por que es cuando la operación módulo 4 vale cero. Y las subclaves intermedias

se calculan directamente aplicando la operación XOR entre el elemento $w[i - Nk]$ y el último elemento calculado, como es el caso de $w[5]$.

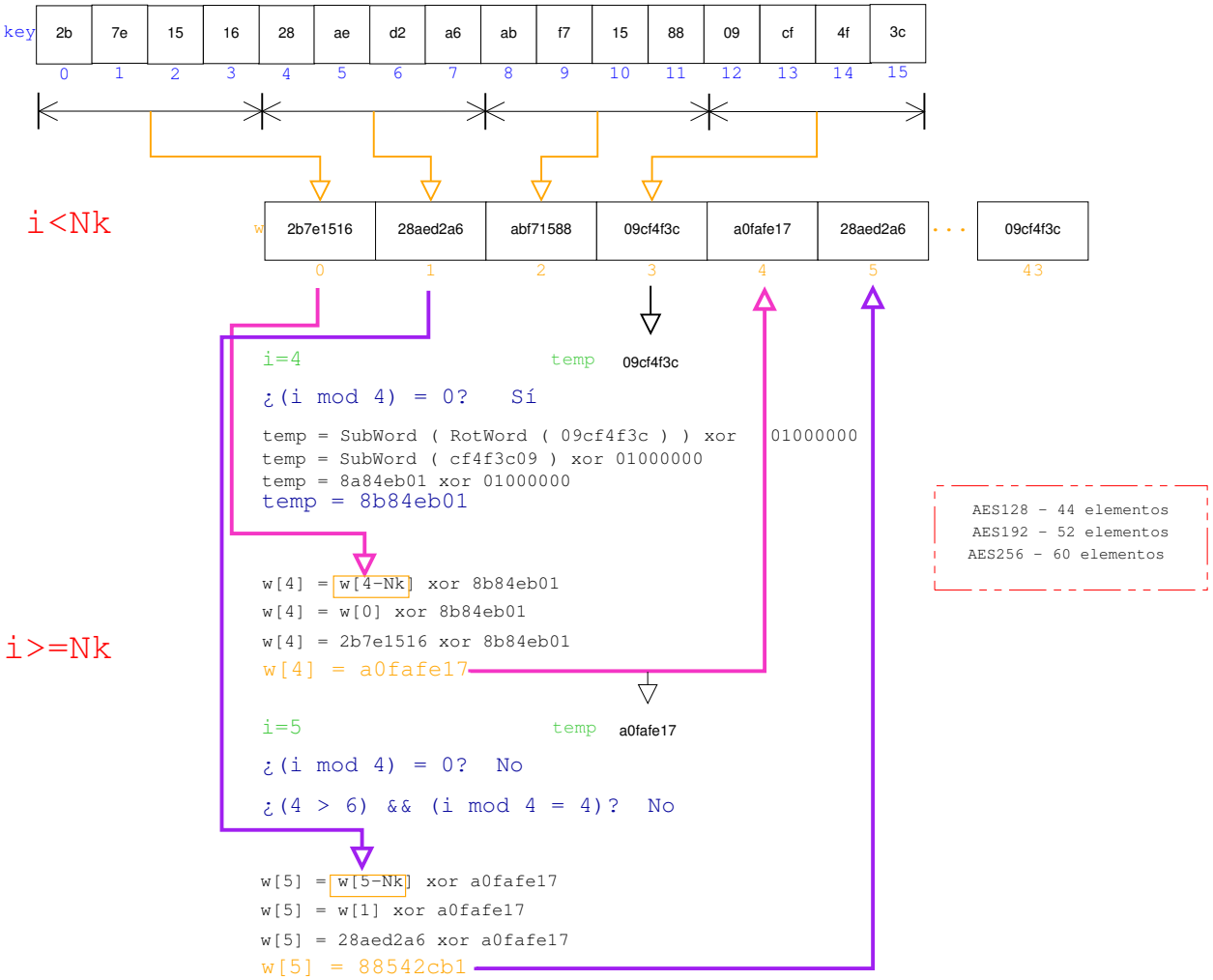


Figura 2.27: Ejemplo de KeyExpansion para una clave de 128-bit [25]

Con los últimos conceptos mencionados se pretende incorporar los conceptos que giran alrededor de un cifrador AES: las teorías matemáticas utilizadas, el algoritmo del cifrador y del descifrador así como sus diferencias, las transformaciones que operan al mensaje y la función que expande la clave dada. Por tanto, lo siguiente a tratar son las herramientas de hardware que se utilizan para implementar un algoritmo de cifrado (Dispositivos Lógicos Programables), así como decir cuando

[25] Clave ejemplo tomada del documento oficial NIST [17]

la implementación de dicho algoritmo se considera un sistema embebido. También se define un IP Core y por ende, la implementación de un IP Core de Cifrado en FPGAs.

2.5 Dispositivos Lógicos Programables

Para poder comprender una implementación hardware, es necesario conocer las opciones existentes de dispositivos lógicos programables, éstos se refieren a aquellos circuitos integrados para aplicaciones embebidas que son programables por un usuario, y se clasifican según su arquitectura hardware:

FPGA (Fiel Programmable Gate Array) Dispositivo de gran capacidad que se forma por un arreglo de celdas básicas programables que están rodeadas por canales horizontales y verticales que los interconectan. Su unidad básica se llama CLB (Configurable Logic Block) por lo que se compone de una matriz de bloques lógicos configurables. Los FPGAs se pueden reprogramar después de su fabricación.

ASIC (Application-Specific Integrated Circuit) Circuito integrado compuesto de componentes electrónicos (transistores, capacitores y resistores), usualmente fabricado sobre una superficie de silicón u otro material semiconductor. En general, es un circuito integrado para funcionalidades dedicadas.

CPLD (Complex Programmable Logic Device) Dispositivo que contiene macro-bloques conectados en una malla de interconexiones programables.

GAL (Generic Array Logic) Dispositivos que son programados y borrados de forma eléctrica de tecnología e^2CMOS.

La lista anterior es de caracter general, de modo que se pueda observar las ventajas y desventajas significativas de usar una y otra opción. Por un lado están los circuitos integrados hechos a medida (ASICs) y por otro, aquellos circuitos programables complejos (CPLD y FPGA) y simples (GAL, etc.). Entonces es notorio que un FPGA sobresale de los demás pues rebasa la capacidad de otros circuitos programables por contener arquitectura hardware avanzada, cuenta también con mayor recursos lógicos y existen varios fabricantes (Xilinx, Altera, Actel, Atmel).

2.6 Sistemas Embebidos

Partiendo de que un sistema es un conjunto de elementos que trabajan para un mismo fin, entonces un sistema embebido es un sistema (dispositivo) dedicado a una función o aplicación específica. Por lo tanto, contiene elementos hardware (puede ser un procesador o un microcontrolador), y elementos sotware (aplicación software que realiza tareas) y por supuesto también contiene los periféricos necesarios para alimentar sus entradas y representar sus salidas de información.

Los SoC permiten la implementación del hardware y software de un sistema digital en un circuito integrado configurable, debido a la facilidad de interactuar con el hardware y el software en el mismo dispositivo. Diseñar en un FPGA resulta flexible pues te permite avanzados niveles de abstracción que son posibles de describir dentro de las herramientas de software específicas. El desarrollo de sistemas embebidos modernos demanda la implementación de funciones sofisticadas en plazos cortos de diseño, cualidad que los dispositivos FPGA posibilitan, además de una gran flexibilidad en el diseño e implementación.

2.7 IP Cores

Un IP Core (Núcleo de propiedad intelectual) es una especificación hardware que puede ser usada para configurar los recursos lógicos de un FPGA u otro dispositivo, y existen dos tipos: hard IP y soft IP. Un soft IP Core permite al usuario modificar alguna configuración y depende del nivel de personalización, que va desde dejar parámetros personalizables en el IP Core, hasta tener el IP Core con su código HDL para que se pueda modificar antes de sintetizar, con la finalidad de saber la funcionalidad y la cantidad de recursos hardware usados en el dispositivo. Y un hard IP Core no puede ser personalizado de ninguna forma, sin embargo provee de datos de rendimiento y el área de hardware que ocupa.

El diseño de propiedad intelectual está teniendo un gran crecimiento en la industria y usuarios y diseñadores están cambiando su infraestructura e interfaces estándar para esta nueva industria. De manera tradicional, los sistemas han sido implementados sobre circuitos impresos usando ICs (Integrated Circuits) de diferentes vendedores. Éstos chips contienen el IP (Intellectual Property) de la compañía que los produce, y al mismo tiempo, los FPGAs fueron usados para implementar

interfaces de diferentes ICs y funciones lógicas simples a la par de las restricciones de su pequeño espacio. Desde entonces, las tecnologías de proceso más pequeñas para ICs har logrado grandes densidades que permiten que la mayoría de los sitemas se implementen en uno o dos chips. Ahora, el diseño de SoC es considerado la tendencia de futuro donde un chip tiene al menos un núcleo microcontrolador programable y memoria.

2.8 IP Cores de Cifrado en Dispositivos Lógicos Programables

El cifrado por software es relativamente lento al consumir grandes recursos del procesador. Si la cantidad de datos a cifrar y/o descifrar no es muy grande y el tiempo de respuesta no es un factor determinante, puede utilizarse como primera opción. Y si por el contrario, si la información es crítica, la cantidad de datos a procesar es considerable y el tiempo de respuesta es importante en el proceso, la mejor opción es utilizar criptografía basada en hardware.

Los FPGA's son inherentemente dispositivos flexibles que pueden ser configurados para implementar cualquier sistema, incluyendo embebidos si es necesario. Éstos dispositivos también pueden ser reconfigurados en la frecuencia que se necesite, por tanto, ofrecen mayor flexibilidad que los ASIC's.

Los algoritmos de criptografía son clasificados por su velocidad de cifrado y descifrado de datos y por su robustez para resistir ataques. El procesamiento de datos de cifrado-descifrado en tiempo real es escencial en aplicaciones basadas en red para seguir el ritmo de tasa de transferencia de los datos de entrada.

Debido a que el desarrollo de éstos algoritmos criptográficos poseen un alto grado de paralelismo, tanto un FPGA como las unidades de procesamiento de gráficos (GPU) son una buena alternativa para ser utilizados como coprocesadores criptográficos para seleccionar diferentes algoritmos de cifrado.

El Centro de Investigación, Innovación y Desarrollo en TElecomunicaciones (CIIDTE) cuenta con tarjetas de desarrollo ZedBoards de la compañía ATMEL, el cual tiene integrado un SoC Zynq 7Z020 de la compañía Xilinx y que tiene las siguientes características (ver Fig. 2.28):

- Sistema de procesamiento (PS). Incluye una Unidad Procesadora de Aplicaciones (APU), interfaces periféricas de entrada/salida, memoria caché, interfaces de memoria y la circuitería que genera el reloj del sistema. La primera contiene dos procesadores ARM Cortex-A9 que operan hasta 1 GHz.

- Lógica Programable (PL). Predominan los elementos de lógica de propósito general (FPGA), que a su vez se compone de conjuntos de CLBs (Configurable Logic Blocks), así como bloques de entrada/salida para poder interactuar .

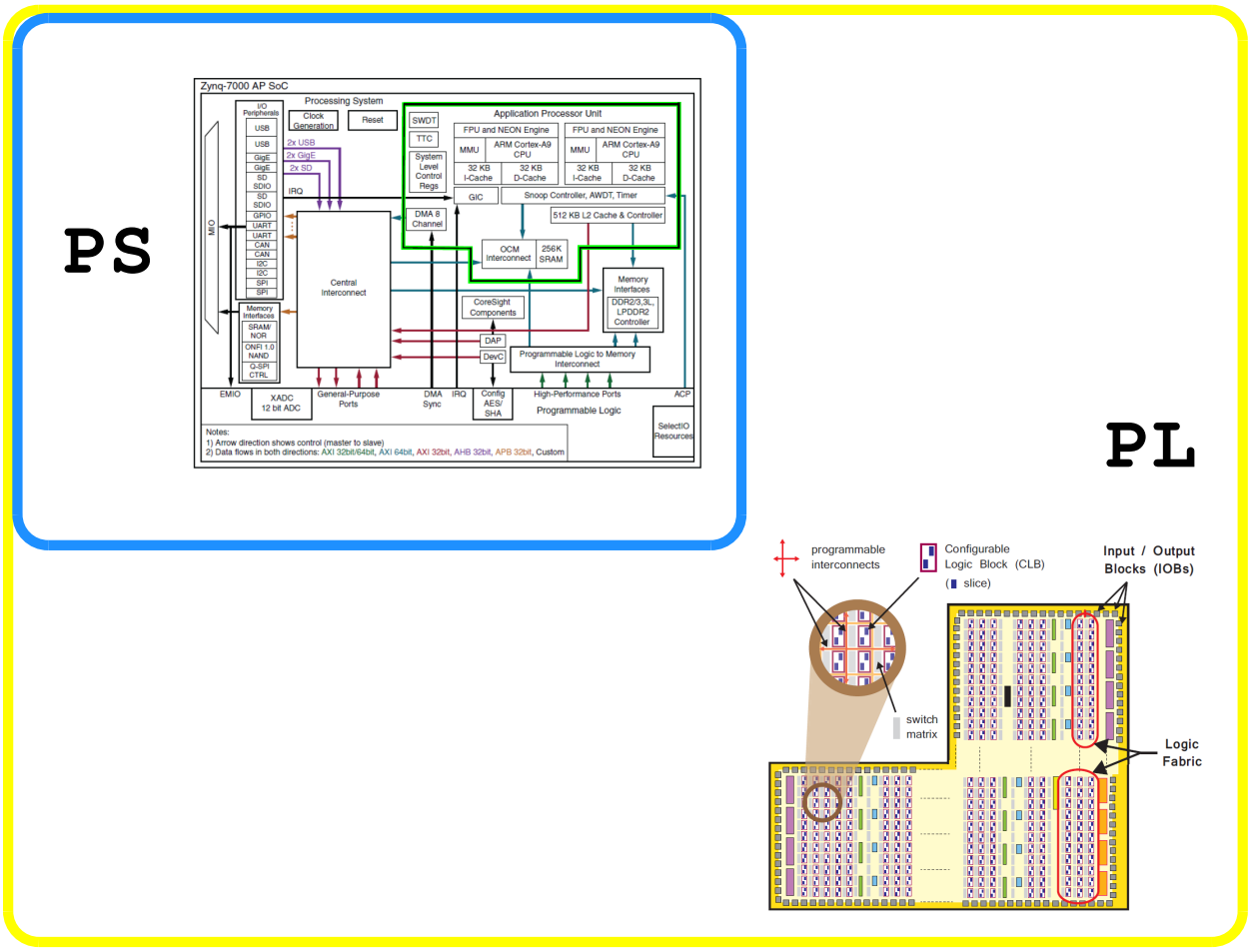


Figura 2.28: SoC Zynq [26]

[26] Imágenes tomadas del documento The Zynq Book [18]

Hoy en día los dispositivos lógicos programables debido a sus características de memoria, reprogramación y paralelismo, los hace candidatos viables para implementar sistemas embebidos [25] en las telecomunicaciones. Éstos últimos también proveen un alto nivel de seguridad al consumir los recursos mínimos[26].

Así, facilmente se puede hablar de un sistema embebido dentro de un FPGA que se encargue de la confiabilidad de los mensajes transmitidos por un sistema de telecomunicaciones, que como tienen gran influencia en todos los aspectos de la cotidianidad, hace que las comunicaciones seguras cobren particular importancia .

2.8.1 Protocolo AXI

Un sistema embebido que hace uso de un dispositivo lógico programable puede albergar más de un IP Core para su funcionamiento, por lo tanto, los IP Cores deben tener la capacidad de transmitir datos siguiendo un protocolo estándar que permita a cada IP Core comunicarse con otros IP Cores. El protocolo AXI (Advanced eXtensible Interface) forma parte de la familia ARM AMBA, y define una interfaz de comunicación entre IP Cores. Existen tres variantes que se diferencian por su funcionalidad y requerimientos del sistema que los use:

AXI4 Full. Direccionamiento tradicional basado en memoria / Interfaz de datos. También soporta ráfagas de datos.

AXI4 Stream. Sólo soporta ráfagas de datos (ver Figura 2.29).

AXI4 Lite. Direccionamiento tradicional basado en memoria / Interfaz de datos. Sólo soporta dato por dato.

En ésta tesis se hace uso del protocolo AXI4 Stream, por que una transmisión de información en un sistema de comunicación requiere de un alta velocidad de transferencia. El estándar AXI4 Stream permite un tamaño de bloque de datos ilimitado, y como así como es compatible con Cores de Xillinx, también lo es con una gran lista perteneciente a la familia ARM. Las señales principales en AXI4 Stream son:

1. tdata. En ella viaja la información que es procesada en cada IP Core.

2. tvalid. Señal que indica que un IP Core maestro tiene un dato válido para ser enviado.

3. tready. Señal que indica que un IP Core esclavo está listo para recibir un dato válido.

4. tlast. Señal que indica que el último dato de la información ha sido transmitido.

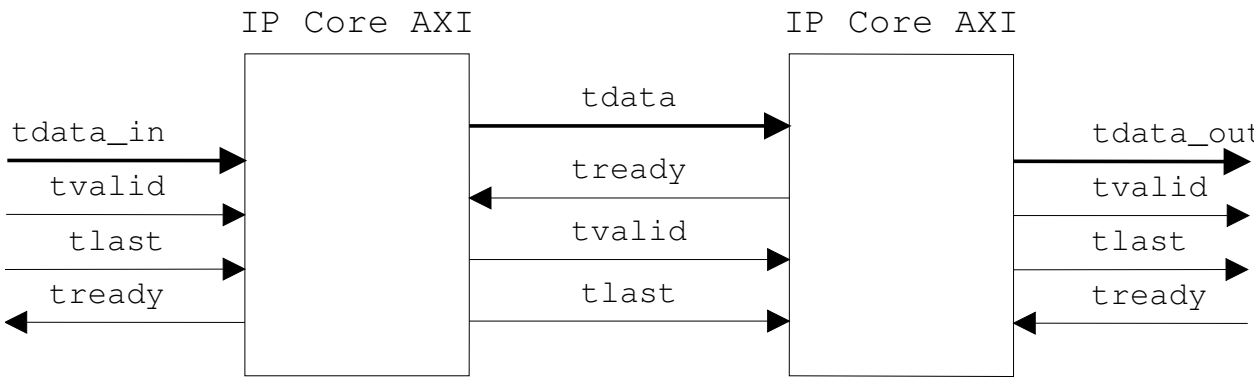


Figura 2.29: Diagrama de Bloques entre IP Cores AXI (maestro-esclavo)

La manera en que opera la interfaz AXI4 Stream hace referencia a la toma de dos roles según sea el caso, es decir, cuando un IP Core recibe una ráfaga de datos se está comportando como un esclavo, y cuando la envía es un IP Core maestro. Entonces, los datos serán transmitidos siempre y cuando el IP Core que envía (maestro) tenga datos válidos y el IP Core que los recibe (esclavo) esté listo para dicha tarea (ver diagrama de tiempo del protocolo AXI Fig. 2.30). De este modo, la información viaja a través de un sistema, lo que convierte a este protocolo en una pieza importante si se quiere reutilzar algún módulo hardware (IP Core).

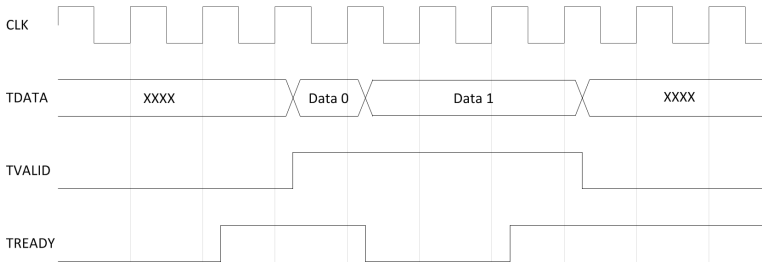


Figura 2.30: Diagrama de Tiempo del Protocolo AXI4 Stream [27]

[27] Imagen tomada del documento UG761 [27]

Hasta ahora se ha terminado de describir los elementos teóricos elementales que intervienen en el desarrollo de un IP Core de Cifrado de Datos AES. En los capítulos posteriores empezamos por revisar las implementaciones hardware de cifrado de datos en otras partes del mundo sobre dispositivos lógicos programables, y así poder comparar el IP Core AES desarrollado en ésta tesis, que también ocupa un capítulo en el documento (ver Capítulo 4). El último capítulo muestra los resultados del algoritmo AES-128 implementado en un SoC (ver Capítulo 5).

Capítulo 3

Estado del Arte

El poder computacional actual permite realizar implementaciones de métodos matemáticos en cualquier PC. Específicamente en criptografía se recomienda utilizar claves de longitud cada vez mayor para que el tiempo de proceso de un cripto analista sobre un criptograma se extienda, y talvés, cuando se logre obtener el dato original no tenga la misma utilidad.

El cifrado por software es relativamente lento al consumir grandes recursos del procesador. Si la cantidad de datos a cifrar y/o descifrar no es muy grande y el tiempo de respuesta no es un factor determinante, puede utilizarse como primera opción. Y si por el contrario, la información es crítica, la cantidad de datos a procesar es considerable y el tiempo de respuesta es importante en el proceso, la mejor opción es utilizar criptografía basada en hardware.

3.1 Cifrado sobre dispositivos Lógicos Programables

En general, el hardware provee alta seguridad, y según [19],la única desventaja que se puede presentar sería escoger indebidamente la longitud de la clave para cuando se manufactura el dispositivo que contiene el cifrador por que no puede ser cambiada despues. Por lo que se ha implementado de muchas maneras algoritmos de cifrado para diversas aplicaciones (ver 3.1). Por ejemplo, [21] hicieron uso del paralelismo para implementar dos algoritmos de cifrado de bloque para una aplicación de análisis. Así mismo, para algoritmos de cifrado asimétrico, se enfocan en optimizar algún bloque clave, por ejemplo [23] y [24] han optimizado el diseño y rendimiento de la multiplicación polinomial y [35] tiene un alto rendimiento en la implementación de campos

finitos. Para el algoritmo AES también existen ese tipo de implementaciones, [22] propone una arquitectura hardware eficiente para la transformación *MixColumns()* e *invMixColumns()*.

La apuesta por modificar alguna funcionalidad sin afectar la estrucura de algún algoritmo es tendencia en las implementaciones hardware para un cifrado AES. Por ejemplo, en [26] se usa una *S-Box* adicional para generar una segunda clave. En [20] el algoritmo AES implementado en FPGA, se hacen variaciones al algoritmo, de forma que en cada transformación la función es diferente a la del estándar pues la *SBox* es distinta, los corrimientos y los polinomios también.

Cuando se trata de realizar módulos hardware en el que se vean implementados más de un algoritmo de cifrado, [36] provee de una familia de cifrados de bloque que son modificables y que por lo mismo, tienen una implementación compacta. En [37] vemos un IP Core con AES256 y TDES con interfaz AXI, mientras que en [38] se tiene de igual manera un acelerador hardware que se utiliza para delegar trabajo al procesador en un sistema de almacenamiento en la nube.

Tabla 3.1: Aportaciones a la Implementación de Algoritmos de Cifrado sobre Hardware.

Año	**Autores**	**Aportación**
2013	Vivek Venugopal-Devu y Manikantan Shila [21]	Estudio de los algoritmos TEA/XTEA usando plataformas reconfigurables y multicores(GPU y FPGA).
2013	Sushma R Huddar-Sudhir y Rao Rupanagudi [22]	Proposición de una arquitectura eficiente para mixcolumns() e invMixColumns().
2013	Aydin Aysu,Cameron Patterson y Patrick Schaumont [23]	Implementación la multiplicación polinomial a bajo costo (Criptografía basada en enrejado).
2014	S. Liu, L. Ju, X. Cai,Z. Jia y Z. Zhang [24]	Implementación de una arquitectura hardware de la multiplicación escalar sobre campos finitos de Curvas Elípticas.
2015	P. Swierczynski, M. Fyrbiak, P. Koppe y C. Paar [33]	Detección y modificación maliciosa de un flujo de datos en un FPGA, creado por un algoritmo criptográfico DES,3DES y AES.
2015	Abhiram L., Srioop B., Gowrav L., Punith K. y Manjunath C. [34]	Implementación en FPGA de claves dobles basadas en cifrado AES.
2015	J. Xie,P.K. Meher y Z. Mao [35]	Alto rendimiento de Campos Finitos implementados en FPGA y ASIC.
2016	C. Beierle, et al. [36]	Familia de cifrado de bloques modificable, implementación compacta.
2016	D. Rakanović y R. Struharik [37]	IP Core AES256-TDES con interfaz AXI.
2017	M. Chellam y R. Natarajam [38]	Acelerador Hardware AES128 sobre FPGA.

Con respecto a los artículos antes mencionados, es evidente que las aplicaciones buscan el rendimiento que el hardware puede otorgarles, por lo que el cifrado que antes sólo se manejaba por software, ahora hace uso también de hardware. Y algunos modifican las características del algoritmo buscando hacerlo más fuerte a las vulnerabilidades del entorno, así como unos cuantos implementan un o más algoritmos de cifrado dentro de un módulo hardware, pero que carecen de una interfaz de comunicación estándar que los haga reusables en distintas aplicaciones. En ésta tesis se desarrolla un IP Core de cifrado/descifrado de bloques AES-128/256 que tiene una interfaz estándar AXI que permite su portabilidad, y que puede abastecer de privacidad al sistema.

En implementaciones desarrolladas por autores mexicanos, se encontraron diversas implementaciones hardware específicamente del algoritmo AES, entre ellas [39], donde se implementa el estándar AES en el modo de operación CBC (ver la subsección 2.3.3), en dicho artículo se desarrolló un AES iterativo con el mínimo de recursos y sin "pipeline" dando como resultados un mínimo uso de LUTs y registros, un Throughput de 1.45 $Gbps$ acorde a los requisitos de la aplicación final del cifrador (> 1 $Gbps$), el costo por lo anterior es utilizar una frecuencia no tan alta. Otro desarrollo [40], es un producto de una tesis donde se implementa un oscilador caótico a modo de cifrador en FPGAs, por lo que en las pruebas en una red de comunicación de 4 elementos con 4 FPGAs se transmitió una señal que a simple vista no es la original. Por lo tanto, el oscilador cumplió su objetivo, donde además es evidente que un dispositivo lógico programable es eficáz a la hora de minimizar recursos y de paralelizar un algortimo matemático. Un último desarrollo resaltable es el artículo [41], donde se implementó de igual manera en un FPGA para redes de computadoras donde se necesita $Gbps$ de procesamiento como ya había mencionado, un algoritmo MD5, que si bien no cifra información, si crea un resumen o huella de la misma. Lo resaltable fue que fueron adaptando sus diseños para lograr el desempeño esperado. Los rasgos similares en comparación con éste trabajo es el diseño en módulos y el futuro uso en un sistema de comunicaciones, sin embargo la implementación solo cubre un AES-128.

Haciendo la revisión del estado del arte se observan variadas implementaciones de cifrado sobre dispositivos lógicos programables, tanto de cifrado simétrico como asimétrico, sin embargo se centran principalmente en arquitecturas hardware que les permite disminuir recursos o aumentar características como autenticidad en el proceso de cifrado, sin embargo no elaboran un IP Core con

un protocolo de comunicación estándar. En [37], se tiene un core que cuenta con una interfaz AXI para dos algoritmos de cifrado de bloques, AES y TDES, dónde la única desventaja es el número de recursos que necesita para su implementación del tipo *pipeline* que supera en mucho a la misma implementación de ésta tesis. Por lo tanto, se estará comparando con el IP Core AES-128/256 con interfaz AXI de ésta tesis, que también se implementó en un SoC *Zynq7000*.

Capítulo 4

Desarrollo

En éste capítulo se describe cómo se llevó a cabo la implementación del algoritmo AES en un FPGA. De tal manera que coincida en gran parte con una metodología de trabajo previamente elaborada.

4.1 Metodología

Para la elaboración de esta tesis, se llevó a cabo la siguiente metología:

(i) Capacitación. Se asistió a cursos que permitieron aundar en el cifrado de datos y en las implementaciones hardware.

(ii) Conceptualización. En ésta etapa se buscó información de desempeño en hardware de algoritmos de cifrado y se optó por implementar el estándar AES-128. Después se hizo una revisión del estado de la cuestión sobre las implementaciones hardware del cifrador AES.

(iii) Módulo Hardware. En ésta etapa se seleccionaron las herramientas de hardware-software para realizar una cama de pruebas que permite verificar el funcionamiento de cualquier IP Core AXI.

(iv) Diseño Hardware. En ésta etapa se realizaron los diseños del cifrador y del descifrador AES que pudiera trabajar con claves de 128/256-bit. Dichos IP Cores se diseñaron también con la posibilidad de comunicarse entre ellos y con otros IP-Cores mediante el estándar AXI4-Stream.

(v) Codificación. En ésta etapa se realizó la codificación en el lenguaje de descripción de hardware Verilog tanto del IP Core de Cifrado AES como del IP Core de Descifrado AES.

(vi) Validación. En ésta etapa se simularon ambos IP Cores por separado con la herramienta ISE Xilinx 14.7.

(vii) Verificación. En ésta etapa se verifico el funcionamiento de los IP Cores conectados entre sí dentro de la cama de pruebas, para observar el cifrado de un bloque de 128-bit seguido del descifrado del mismo bloque y obtener el dato original.

4.2 Diseño de Hardware

El diseño de hardware comprende tanto al algoritmo de cifrado AES como a la interfaz AXI, y éstos en conjunto se fusionan en un módulo hardware al que llamamos IP Core AES. En el algoritmo de cifrado AES se destacan dos tareas: la expasión de la clave ($keyExpansion()$) y la codificación como tal. Dicha codificación o cifrado AES se diseñó partiendo de las distintas necesidades que pudiera tener un sistema, por lo que se crearon tres versiones: un diseño combinacional, un diseño pipeline y un diseño secuencial, cada uno con sus respectivas ventajas y desventajas.

Entonces, y de primera instancia, se muestra en la Fig. 4.1 el conjunto de entradas y salidas del IP Core AES, donde las señales de color azul corresponden al control de flujo de datos (AXI4-Stream), las señales de color lila corresponden a la configuración inicial del IP Core (el tipo de AES-128/256, la clave privada y la carga de la clave al IP Core) y las señales de color negro sólo son dos banderas indicativas del momento en que la clave ha sido expandida y del momento en que el cifrado ha finalizado.

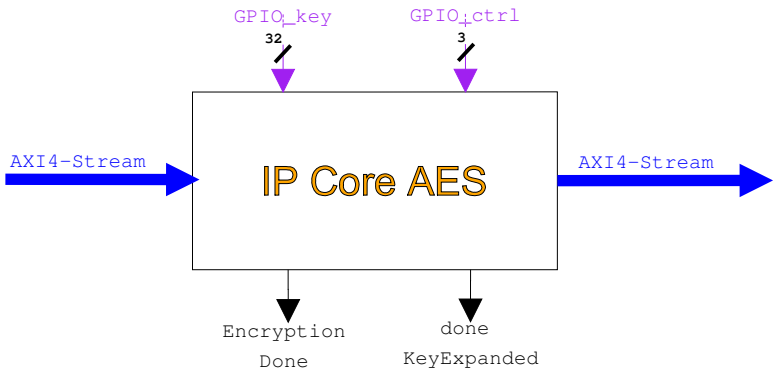


Figura 4.1: IP Core AES. Bloque de entradas y salidas

Para realizar un cifrado/descifrado de datos (un bloque 128-bit por ejemplo), lo primero que se tiene que realizar es la configuración del tipo del AES mediante la entrada $GPIO\_ctrl$, donde los dos bits menos significativos se modifican según el AES requerido y el bit más significativo indica la carga para cada 32-bit de la clave (ver Fig. 4.2). Por ejemplo, para un cifrador/descifrador AES-128 que utiliza una clave de 128-bit se tendrán cuatro cargas y ese bit tendrá cuatro pulsos positivos para cargar la clave completa. Entonces, ya configurado el IP Core y cargada la clave, los datos a cifrar/descifrar son transmitidos vía AXI cada 8-bit y son entregados cada 8-bit con éste mismo protocolo.

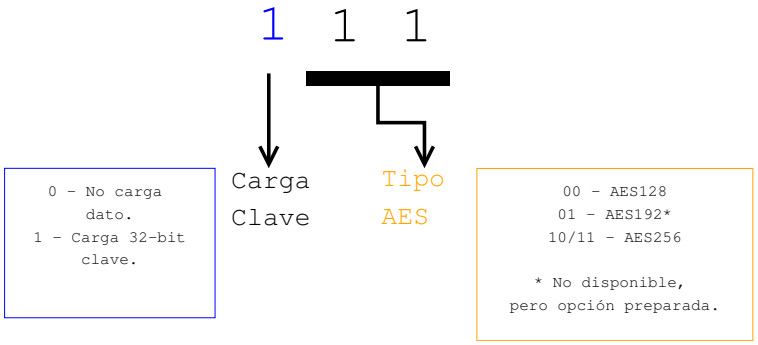


Figura 4.2: Configuración del tipo de AES y Carga de la clave: $GPIO\_ctrl[2:0]$.

Viendo el diagrama de caja blanca del IP Core, nos encontramos con los elementos hardware que lo componen (ver Fig. 4.3) : a) La rutina de la clave ($KeyExpansion()$) y la memoria donde se almacena la clave expandida, b) El algoritmo de cifrado AES y c) Los módulos hardware AXI4-Stream.

Cuando se configura el tipo de AES y se carga la clave, el módulo $KeyExpansion()$ expande la clave y la guarda en una memoria RAM, mandando un pulso llamado $doneKeyExpanded$ al finalizar. Después, al iniciar la transmisión de los 128-bit de datos por la interfaz AXI (recordando que el algoritmo recibe 128-bit y entrega 128-bit cifrados al igual que el descifrador) byte a byte, sólo si *tvalid2master* es igual a 1 y hasta que *tlast_from_master* se igual a 1 y es entonces cuando el cifrador recibe el dato en su totalidad para que el algoritmo lo cifre. Éste último utiliza partes de la clave expandida que consulta en la memoria RAM para las rondas que lo componen. Cuando el algoritmo ha terminado su función, la señal *EncryptionDone* es igual a 1, por lo que ya listos los 128-bit cifrados, son enviados byte a byte sólo si la señal *tready_from_slave* de un core esclavo es

igual a 1 y hasta que la señal *tlast2slave* también sea 1. De una manera similar opera el IP Core de Descifrado, en lo que respecta a la configuración inicial y al descifrado de 128-bit y entrega del dato descifrado vía AXI4-Stream es el mismo procedimiento, el cambio se lleva a cabo dentro del algoritmo, y que se verá a detalle en las siguientes subsecciones.

El diseño del algoritmo de cifrado y descifrado respectivamente, se llevó a cabo en tres versiones y éstas son descritas en la subsección 4.2.2, así como el diseño de la rutina de la clave que irá en ambos IP Cores (difrado/descifrado). Por su parte, los módulos AXI4-Stream fueron parte de un trabajo previo [30], por lo que sólo se usaron los diseños existentes.

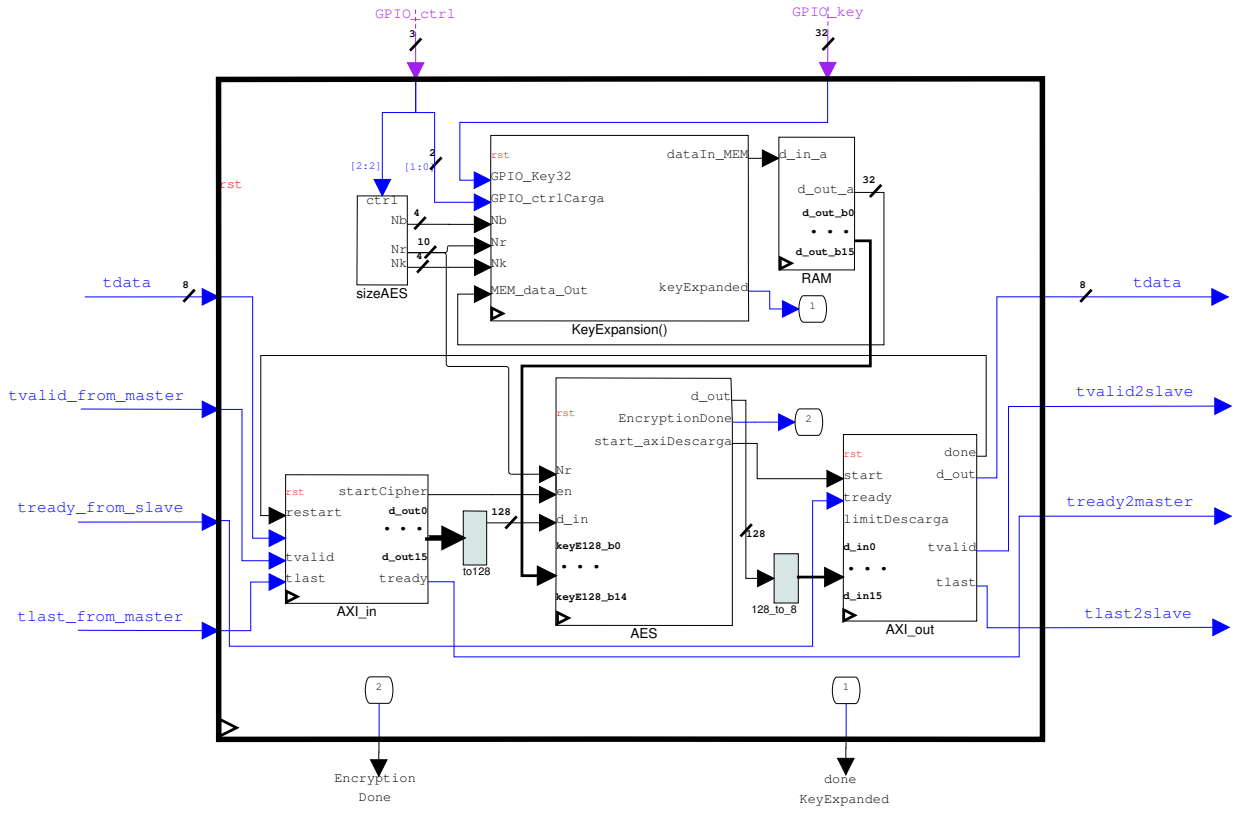


Figura 4.3: IP Core AES. Bloque de entradas y salidas y sus componentes hardware.[1]

[1]El diseño tiene un reloj y reset común al sistema.

4.2.1 Rutina de la clave ($keyExpansion()$)

El diseño de la rutina que expande la clave se basó en que dos acciones principales: la carga de clave y el cálculo de los elementos restantes de la clave expandida.

Como se vió en la sección 2.4.2.5, los primeros elementos de la clave expandida corresponden a la clave dada, y por eso el primer paso es cargar la clave original a las primeras localidades de una memoria que contendrá la clave expandida al finalizar la rutina. La clave es cargada cada 32-bit en la entrada **GPIO_Key32** siempre y cuando la rutina reciba también un pulso positivo en el MSB de la entrada **GPIO_ctrlCarga**. Una máquina de estados es la responsable de cargar la clave (**FSM_carga** ver Fig. 4.4), cuando la tarea está lista manda un pulso indicando que terminó (done=1).

La siguiente tarea es completar la clave expandida, por lo que la señal de término de carga de clave es ahora la señal de comienzo para la máquina de estados que permite el cálculo de los elementos restantes (**FSMGeneraW**). Para calcular cada nueva subclave se necesita la última calculada y aquella Nk posiciones antes. Entonces, cuando **FSMGeneraW** termine, también mandará una pulso (**KeyExpanded**) que nos indica que la clave está expandida y almacenada en memoria (**mem_tamVar**).

Ahora es posible describir la conexión entre los bloques del diseño de la Fig. 4.4, donde los elementos corresponden a los del algoritmo del estándar (ver subsección 2.2.5), entonces las máquinas de estado antes mencionadas, controlan mediante sus señales de salida el flujo de la clave expandida. **FSMGeneraW** necesita que **FSM_carga** termine su proceso de carga primero, después es cuando realiza diversas operaciones sobre el fragmento de clave que se esté operando. Ambas acceden a memoria, la primera guarda la clave cargada, y la segunda la consulta para generar la clave expandida completa, por lo que utilizan multiplexores para no acceder a memoria en el mismo intervalo de tiempo. Por su parte, existen registros temporales (**temp0 y temp1**) que almacenan el dato y que sea procesado sin modificar el original. En general, cada módulo del diseño funciona según el algoritmo, sólo es importante mencionar que la tabla de **Rcon** es guardada dentro de un registro con el mismo nombre, de modo que no se tenga que calcular. Todos los procesos están condicionados al tipo de clave que se haya escogido con el lsb de la seña **GPIO_ctrlCarga**,

y al finalizar cuando **KeyExpanded = 1**, en las salidas **keyE128_b0 ... keyE128_b15** estará almacenada la clave expandida generada.

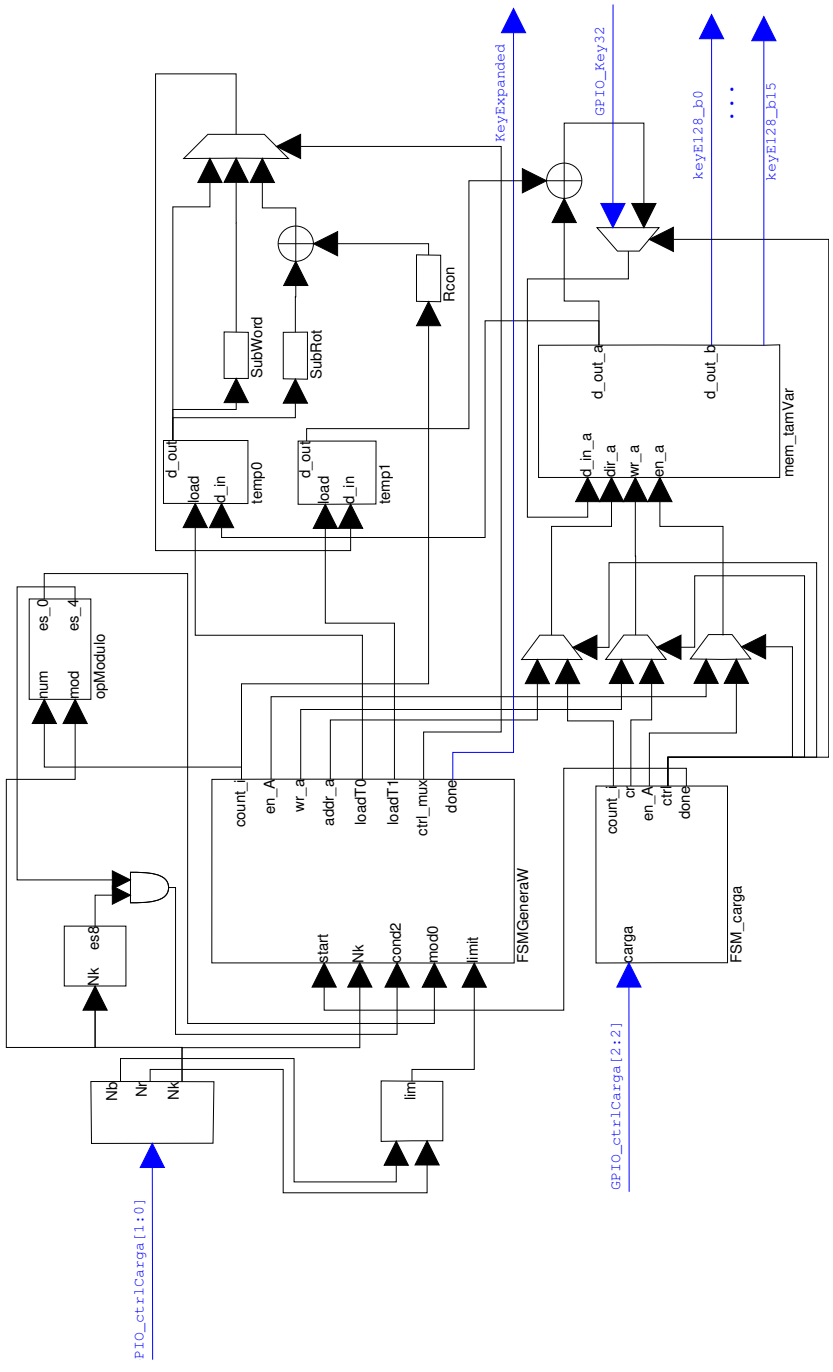


Figura 4.4: Diseño de la rutina de la clave $keyExpansion()$

4.2.2 Cifrador AES

ă ă El algoritmo de cifrado AES permite realizar diseños según requerimientos de recursos, frecuencia máxima, etc. En este documento se diseñaron tres[2] arquitecturas hardware AES y la rutina de clave que se encarga de expandirla. Para poder armar una arquitectura hardware del cifrado AES, fue necesario diseñar como primera instancia cada una de las cuatro transformaciones, y a partir de ahí decidir la arquitectura del algoritmo AES.

El primer diseño es del tipo combinacional (ver Figura 4.5), de modo que se ejecuta en un ciclo de reloj. La entrada al algoritmo es un bloque de datos de 128-bit que es procesado por la transformación *AddRoundKey()* seguido de un bloque que comprende a *SubBytes()-ShiftRows()-MixColumns()-AddRoundKey()* y que se repite 9 veces, para finalizar con el último bloque que comprende a *SubBytes()-ShiftRows()-AddRoundKey()*.

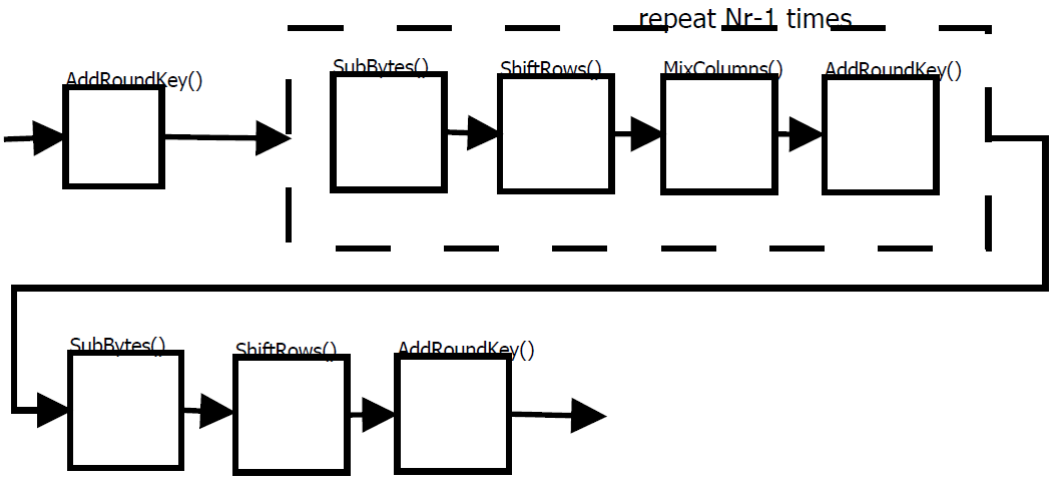


Figura 4.5: Diseño AES Combinacional

[2]combinacional, pipeline y secuencial, donde cada una cuenta con ventajas y desventajas, tanto en uso de recursos, en Throughput y en frecuencia de operación.

El segundo diseño es de tipo *pipeline* (ver Figura 4.7), donde el primer bloque de 128-bit de datos cifrado, es el único que tarda más de un ciclo de reloj en ejecutarse, a diferencia de los siguientes bloques de datos, y es ahí donde radica uno de los beneficios de la concurrencia en un sistema. Por ejemplo, en la Figura 4.6, el primer dato de entrada a la secuencia de funciones $A - B - C - D$ tardará 4 ciclos de reloj en salir, mientras que el segundo dato de entrada tardará sólo un ciclo de reloj.

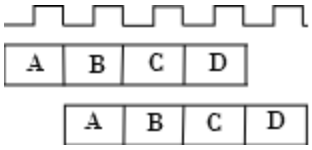


Figura 4.6: Ejemplo de Pipeline.

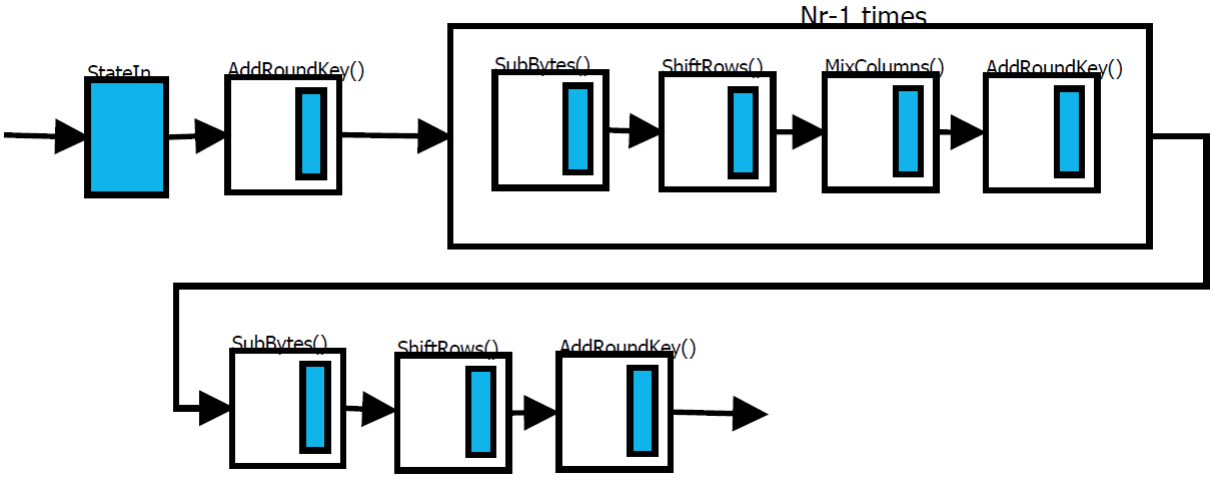


Figura 4.7: Diseño AES Straighforward

El tercer diseño consta de una arquitectura secuencial (ver Figura 4.8), en el cual se usan menos recursos puesto que las transformaciones no se repiten diez veces, si no que se ejecutan cuando se necesitan y hasta entonces son habilitadas. Éste diseño hace uso de un elemento llamado FSM(Finite State Machine), que es un modelo que funciona como un control de ejecución de flujo, por lo que las salidas de una máquina de estados son regularmente señales de control de otros módulos. Una FSM se compone de un conjunto de estados donde cada uno es activado a

la vez, el comportamiento de una FSM puede ser representado con un grafo con distintas condiciones que hace que el sistema habilite o deshabilite algún elemento y/o lleve cuentas. La FSM correspondiente a esta arquitectura hardware se puede observar en la Tabla 4.1, así como una breve descripción de ella.

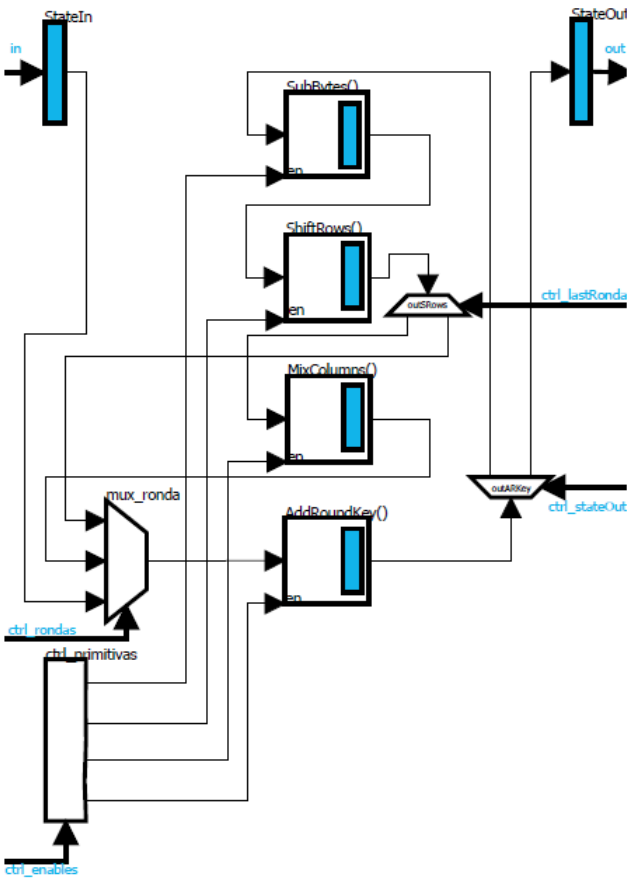


Figura 4.8: Diseño AES Secuencial

4.2.3 Descifrador AES

De igual modo, para el diseño del descifrador, se realizaron tres arquitecturas hardware: una de tipo combinacional, una de tipo pipeline y una última de tipo secuencial. El diseño del descifrador difiere del diseño del cifrador en varios aspectos que resulta en diferencias considerables principalmente en *invMixColumns()* pues contiene distintas ecuaciones, en *invSubBytes()* se utiliza una tabla *S-Box* con distintos valores, en *invShiftRows()* se realizan las rotaciones cíclicas a la derecha y mientras que *AddRoundKey* mantiene su funcionalidad original.

El último diseño corresponde a un descifrador secuencial, un diseño ligeramente diferente al cifrador secuencial, pero que utiliza menos estados en su FSM (ver Tabla 4.2). La máquina de estados es quien se encarga de decirle a cada elemento del diseño cuando debe operar y también lleva una cuenta que permite seleccionar el rango de la clave expandida según se requiera y para que *AddRoundKey()* haga su función.

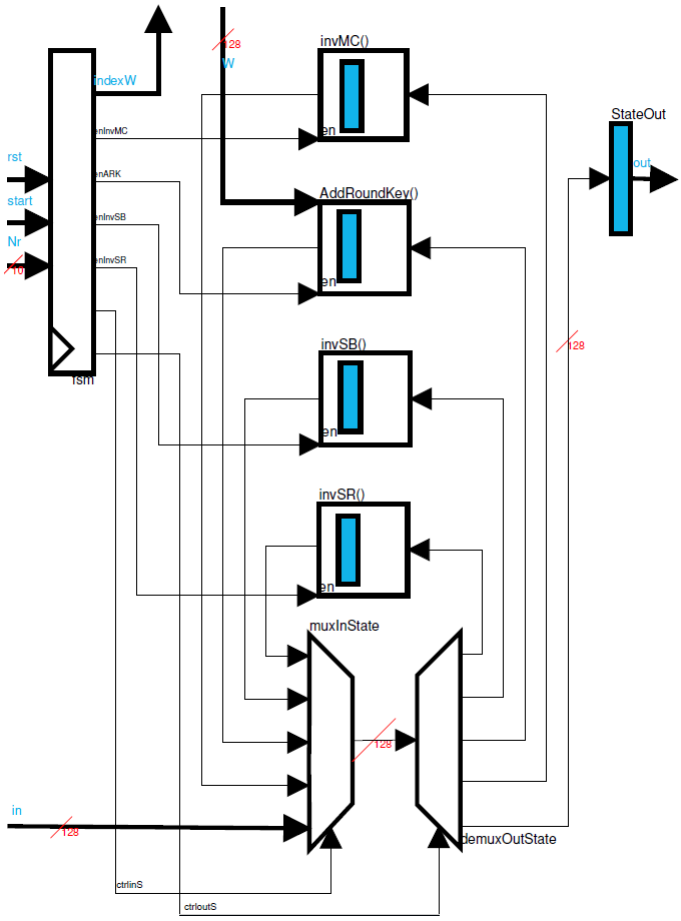


Figura 4.9: Diseño del Descifrador AES

4.3 Descripción de Hardware

La descripción hardware de todos los elementos del cifrador y descifrador AES, se desarrollaron una vez finalizado el diseño de los mismos. El lenguaje usado para la descripción de hardware es conocido como `Verilog`®, y es un estándar aceptado por la IEEE (Institute of Electrical and Electronics Engineers) desde 1995 pero que existía desde 1983. Éste lenguaje HDL (Hardware Description Languaje) es usado para describir circuitos digitales modelando la concurrencia de procesos encontrados en hardware. La sintaxis de Verilog HDL es similar a la del lenguaje de programación C, además de que permite distintos niveles de abstracción (switches, RTL ó código). Para realizar diseños en Verilog, no es necesario especificar el fabricante del FPGA que se está usando y esto permite describir un sistema para cualquier FPGA. Por otro lado, los lenguajes de

verificación como System Verilog, combinan el paralelismo del hardware con el paradigma orientado a objetos de C++ para lograr un alto grado de confianza en la funcionalidad del hardware. En futuros proyectos se desea realizar la verificación de los IP Cores existentes en el lenguaje SystemVerilog, y es por ello que para evitar un letargo en el camino, se describen los sistemas en Verilog.

El IDE que se usó para describir al cifrador AES fue *ISE Design Suite v14.7* de *Xilinx*. Los primeros elementos en ser descritos en Verilog fueron las cuatro transformaciones que conforman al algoritmo AES: *AddRoundKey()*, *SubBytes()*, *ShiftRows()* y *MixColumns()*, para después sólo instanciar dichos elementos según el diseño. En la ausencia de *KeyExpansion()*, ésta fue reemplazada por un arreglo que ya contenía la clave expandida para el caso AES128, para que en unión con el dato de entrada al cifrador, se pudiera observar cada cambio en la matriz *State* al ser procesada por las transformaciones en repetidas ocasiones. Posteriormente se describió la función *KeyExpansion()* y fue agregada al proyecto para prescindir del arreglo antes mencionado.

Una vez creados los diseños secuenciales hardware del cifrador y descifrador, se diseñaron las máquinas de estado que son capaces de controlar a cada elemento hardware involucrado. Entonces con la máquina de estados describir su comportamiento resulta realmente sencillo. Las máquinas de estados diseñadas para el cifrador y el descifrador fueron realizadas en la herramienta de software llamada *Aldec Active-HDL 8.3*, y son mostradas en la Tabla 4.1 y en la Tabla 4.2 respectivamente.

Tabla 4.1: Máquina de estado del Cifrador secuencial AES

Breve descripción
La FSM del **Cifrador AES** está constituida de 12 estados ($S0, ..., S11$), el estado inicial $S0$ es el inicio por defecto (con un reset que es común a toda la arquitectura hw) del cifrador hasta que recibe *start=1* que avanza al estado siguiente $S1$ seleccionando la transformación *AddRoundKey()* con la opción 3 y avanzando al estado $S2$ que recorrerá las cuatro transformaciones en orden ($cta\_a = 4$) y cada que se repita alguna se incrementa en uno el contador cta_b que cuando sea $cta\_b = Nr - 1$ cambiará al estado $S5$. Después se activarán las últimas tres transformaciones del algoritmo, una por cada estado ($S5$ al $S8$). Al final, sólo se envía la señal de término del cifrado con $done = 1$ en $S9$, y en los estados restantes la señal $wr\_memAXIdescarga = 1$ y se da inicio a otra fsm ($start\_axiDescarga = 1$).

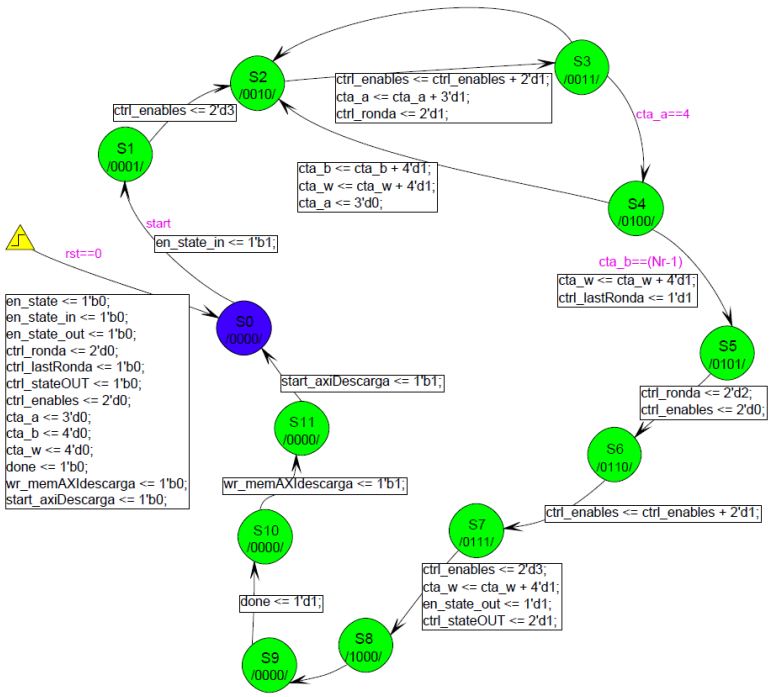
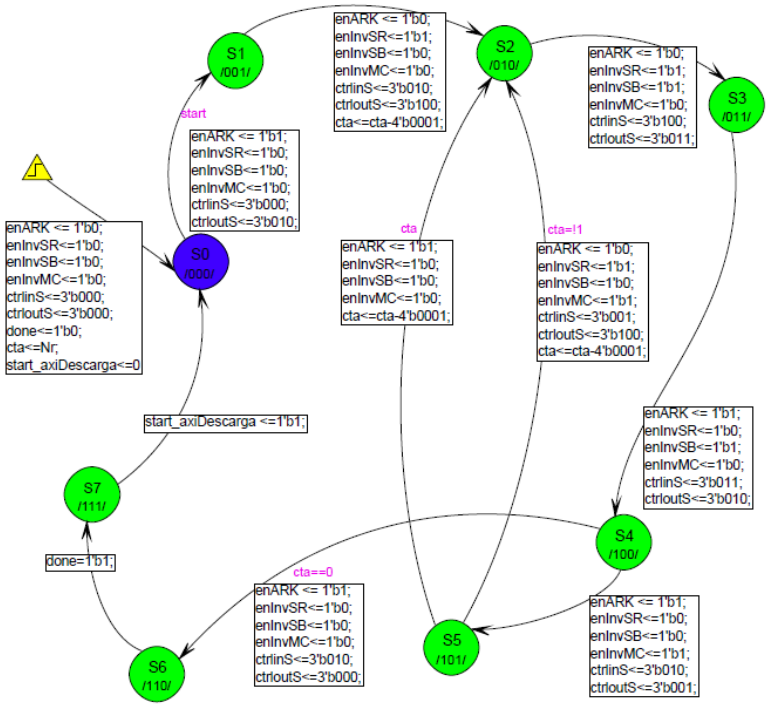


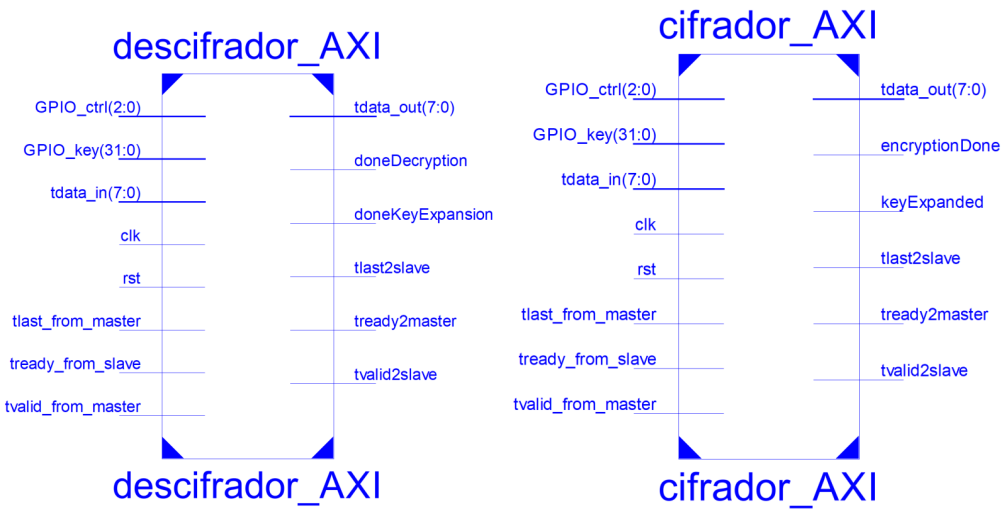
Tabla 4.2: Máquina de estado del Descifrador secuencial AES

Breve descripción
La FSM del **Descifrador AES** difiere de aquella correspondiente al cifrador por que el diseño es diferente (las transformaciones se activan de forma individual y no a través de un decodificador), está constituida por sólo 8 estados ($S0, ..., S7$), el estado inicial $S0$ es el inicio por defecto del cifrador (con un reset que es común a toda la arquitectura hw) hasta que recibe *start=1* para avanzar al estado siguiente $S1$ habilitando la transformación *AddRoundKey()* y luego avanzar sin restricción al estado $S2$ habilitando *invShiftRows()*, y en el siguiente *invSubBytes()* y en el siguiente *AddRoundKey()*, y en el siguiente estado ($S5$) *invMixColumns()* mientras $cta \neq 0$ (que significa que se sigue operando en las rondas intermedias). La última ronda se opera cuando $cta = 1$ y ahí la única transformación que no se habilita es *invMixColumns()*, donde se decrementa en 1 cta para que llegando al estado $S4$ tenga el valor de $cta = 0$ cambie de estado al estado $S6$ y luego al estado $S7$ con la señal de salida $done = 1$ (proceso de descifrado terminado) y posteriormente del estado $S7$ al estado $S8$ se envíe la señal $start_axiDescarga = 1$ indicandole a otra máquina de estados que inicie.



4.4 IP Core AES

El IP Core de cifrado AES se forma del algoritmo AES y del protocolo de comunicación AXI4-Stream, así como el IP Core de descifrado AES se forma del algoritmo de descifrado AES y del mismo protocolo de comunicación. Cada uno de ellos tiene la capacidad de operar en el modo AES-128 y AES-256, y se tiene la opción de configurar también para el modo AES-192 aunque no esté disponible. La configuración para el tipo de AES se realiza cambiando los bits de la señal **GPIO_ctrl[1:0]** como se vió en la sección 4.2.



(a) Cifrador AES AXI4-Stream (b) Descifrador AES AXI4-Stream

Figura 4.10: RTL del Cifrador y Descifrador AES AXI4-Stream

La Fig. 4.10 muestra el RTL (Register-Transfer Level) que es una representación abstracta en alto nivel de un circuito específico, en éste caso de los algoritmos de cifrado y descifrado descritos en *Verilog*. Entonces, la Fig. 4.10a y la Fig. 4.10b son los circuitos que posteriormente fueron añadidos a la plataforma de hardware con las respectivas conexiones entre ellos, donde además de la fuente de los datos y el depósito de los datos procesados, cabe resaltar que el cifrador AES toma el papel de IP Core maestro y el descifrador AES el de esclavo bajo el protocolo *AXI4-Stream*.

En el siguiente capítulo se describen los resultados partiendo de la simulación de un cifrado-descifrado de los IP Cores generados (Fig. 4.10) para un AES-128 y un AES-256. Posteriormente se lleva a cabo la implementación del IP Core AES-128 en la tarjeta de desarrollo *ZedBoard*.

Capítulo 5

Resultados

En este capítulo se muestran los datos arrojados por el IP Core de Cifrado y el IP Core de Descifrado, los cuales son explicados a detalle. Existe un documento oficial llamado *"The Advanced Encryption Standard Algorithm Validation Suite"* [1] que contiene un listado de valores prueba con el correspondiente criptograma resultante, y con el que se corroboró el funcionamiento del IP Core de Cifrado/Descifrado AES-128 en modo ECB. Los valores de prueba están divididos de forma que abarquen todas las combinaciones posibles para asegurar que cualquier dato (128-bit) pueda ser cifrado/descifrado con cualquier clave (128/256-bit) por el IP Core AES-128/256, y para poder corregir los errores tipográficos en la *S-box* si existieran. Para fines prácticos, las capturas de los resultados corresponden a un ejemplo en específico que se encuentra en el Apéndice B del documento oficial [17].

5.1 Simulaciones

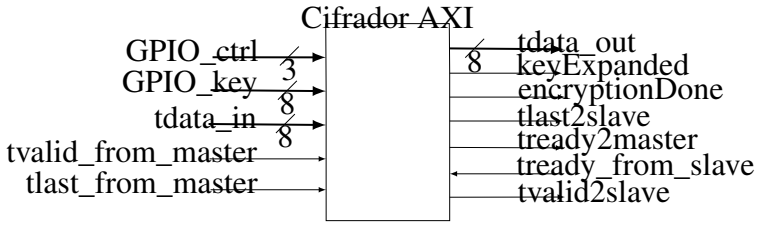
Después de que el algoritmo de cifrado AES-128/256 fuera descrito en Verilog dentro del entorno de ISE Design Suite 14.7 y se le hubiera agregado la interfaz de comunicación AXI4-Stream, el algoritmo fue simulado. Los datos de entrada/salida del algoritmo AES del Cifrador AXI (ver Fig. 5.1a) que se necesitan para obtener el *criptograma* son los siguientes:

(i) GPIO_ctrl [3-bit]: 000.

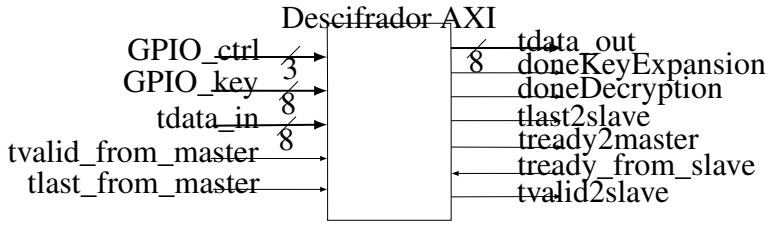
(ii) GPIO_key [32-bit]: **2b 7e 15 16 28 ae d2 a6 ab f7 15 88 09 cf 4f 3c**.

(iii) tdata_in [8-bit]: **32 43 f6 a8 88 5a 30 8d 31 31 98 a2 e0 37 07 34**.

(iv) tdata_out [8-bit]: **39 25 84 1d 02 dc 09 fb dc 11 85 97 19 6a 0b 32**.



(a) Bloque AXI: Cifrador

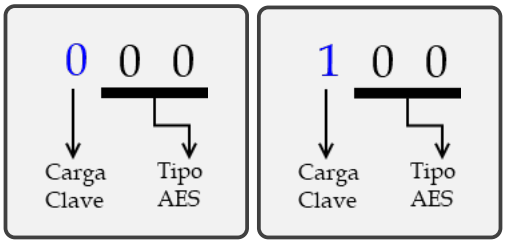


(b) Bloque AXI: Descifrador

Figura 5.1: Bloques AXI: Cifrador-Descifrador

El dato de entrada [i] une a dos señales de control: un bit (MSB) para cargar 32-bit de la clave y dos bits 00-01-10 para escoger el tipo de AES (128-192-256). Independientemente del tipo de AES, la primera tarea del cifrador es cargar la clave desde un GPIO, y como la capacidad de tal periférico es de 32-bit, se realizan varias cargas. Para AES-128 donde la clave es de 128-bit, la señal *GPIO_ctrl* tiene el valor de 0_{10} (Fig. 5.2a) seguido de 4_{10} (Fig. 5.2b) para cargar 32-bit de la clave, por lo que éstos valores se repetirán cuatro veces (ver valores resaltados en rojo en la Fig. 5.3a). Por su parte, para un cifrado/descifrado AES-256, la clave es de 256-bit por lo que se realizarán 8 cargas, y la señal enviada a *GPIO_ctrl* tendrá el valor de $0_{10}/000_2$ seguido de un $6_{10}/110_2$ (ver Fig. 4.2).

La expansión de la clave es el proceso siguiente, y cuando llega a su fin, la señal *keyExpanded* cambia su valor de '0' a '1' indicando que la clave ha sido expandida y que se encuentra almacenada para ser consultada por el cifrador. Después (ver Fig. 5.3b), el cifrador empieza a recibir el bloque de datos de 128-bit en 16 segmentos de 8-bit a través de tdata_in (iii), y al terminar el proceso de cifrado la señal *encryptionDone* cambia su valor de '0' a '1'. El dato de salida *tdata_out* devuelve el bloque de datos cifrado en 16 segmentos de 8-bit como se muestra en la Fig. 5.3c.



(a) GPIO_ctrl (b) Carga 32-bit

Figura 5.2: Dato $GPIO\_ctrl$ para AES128.

De igual modo, el Descifrador AXI (ver Fig. 5.4a) necesita la misma clave dada por el GPIO, el tipo de AES con el cual se cifraron los datos (v) y el criptograma (vii, ver 5.4b), para que devuelva el dato original (viii, ver 5.4c). En un principio, el valor de *donKeyExpansion* cambia de '0' a '1' cuando la clave ha sido expandida; después el valor de *doneDecryption* cambia de '0' a '1' cuando el bloque de 128-bit ha sido recuperado y debe ser igual al bloque original.

(v) GPIO_ctrl [3-bit]: 000.

(vi) GPIO_key [32-bit]: **2b 7e 15 16 28 ae d2 a6 ab f7 15 88 09 cf 4f 3c**.

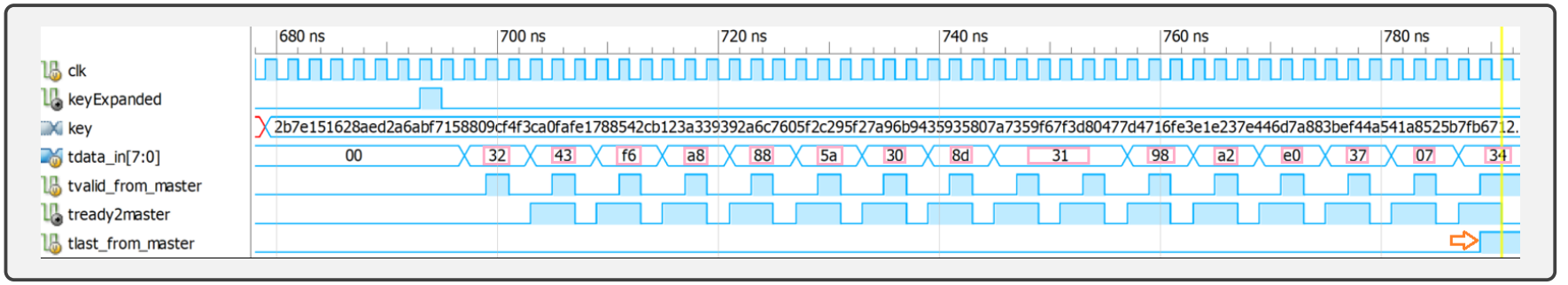
(vii) tdata_in [32-bit]: **39 25 84 1d 02 dc 09 fb dc 11 85 97 19 6a 0b 32**.

(viii) tdata_out [32-bit]: **32 43 f6 a8 88 5a 30 8d 31 31 98 a2 e0 37 07 34**.

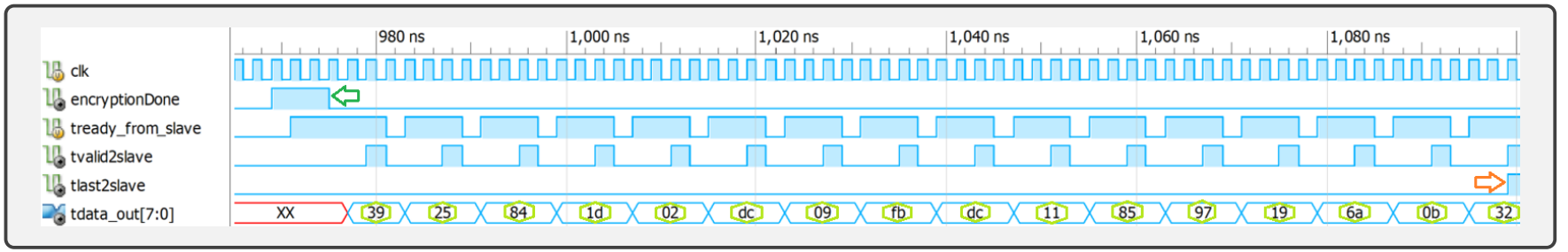
Y sólo a manera de demostración, se muestra la simulación del cifrador AES-256, puesto que también cifra bloques de 128-bit y lo que lo diferencia es el tamaño de la clave, cuantas cargas se realizan para poder expandir dicha clave y el número de rondas que realiza el cifrador (ver Fig. 5.5).

[1] clave de 4-word resaltado en amarillo, bloque de dato de 16-byte resaltados en rosa y criptograma de 16-byte resaltados en verde.

(a) Envío de Clave de 128-bit cada 32-bit

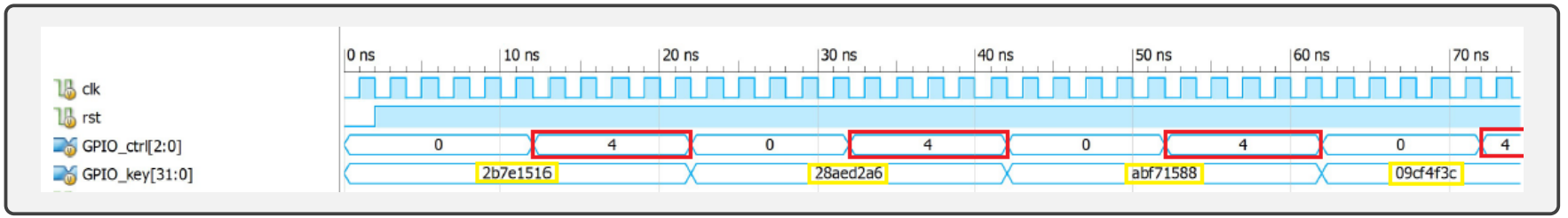


(b) Clave expandida y Dato enviado cada 8-bit

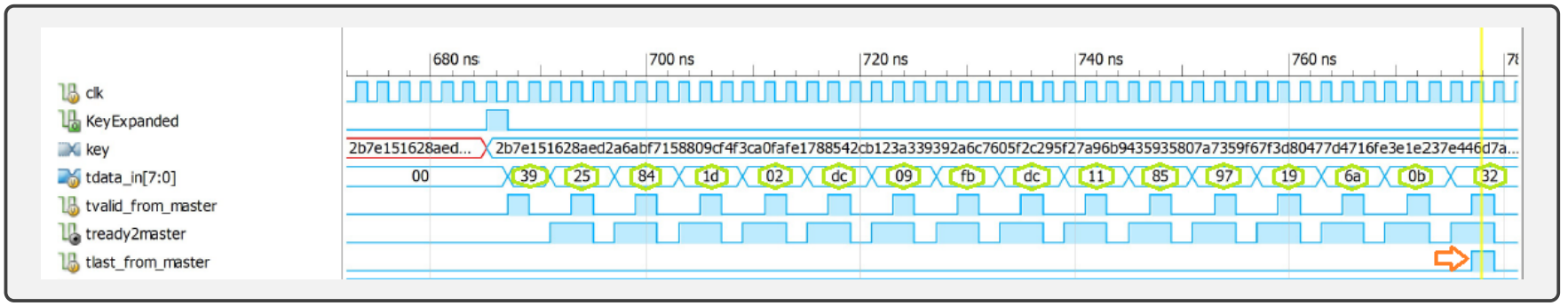


(c) Criptograma

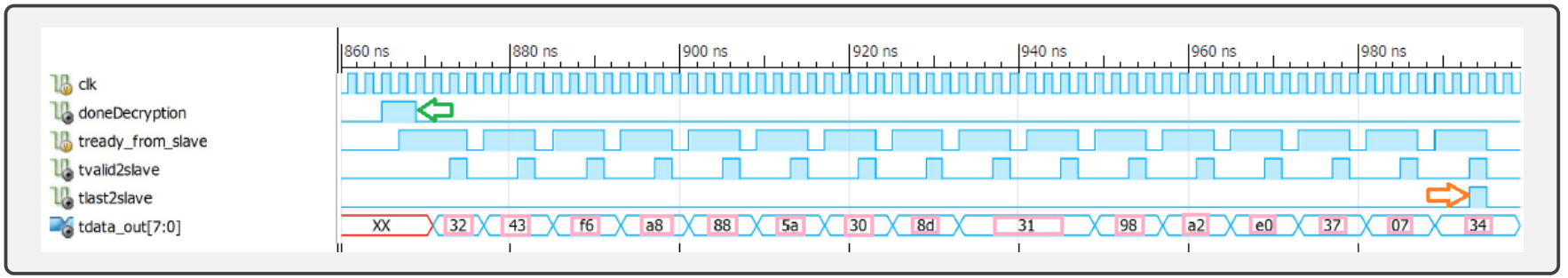
Figura 5.3: Simulación de Cifrador AES-128.



(a) Envío de Clave cada 32-bit

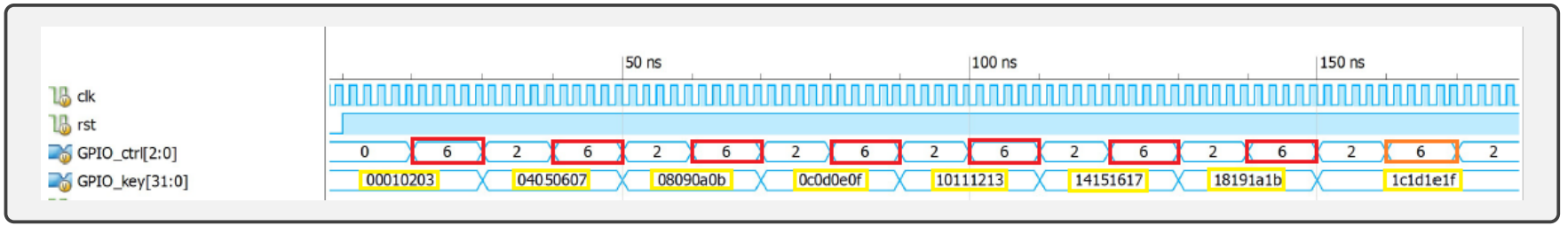


(b) Clave expandida y Criptograma enviado cada 8-bit

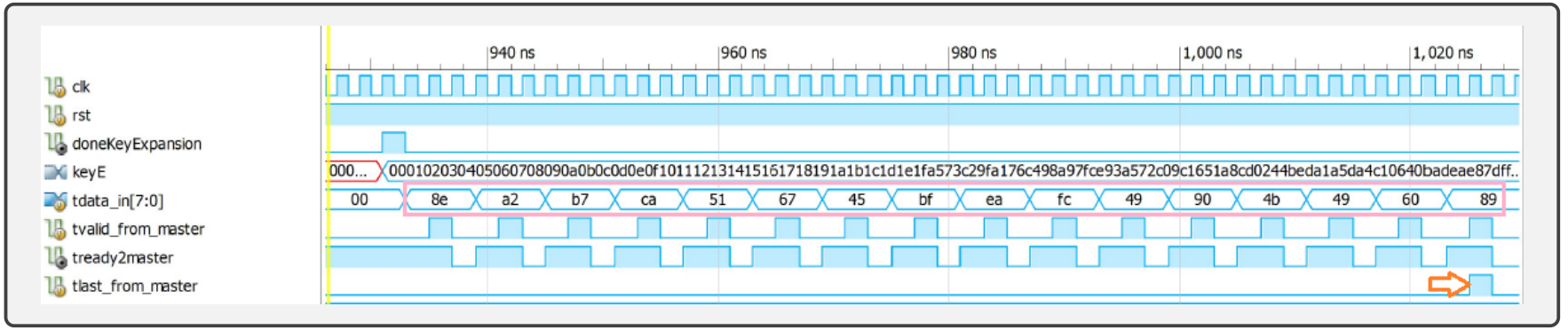


(c) Dato original

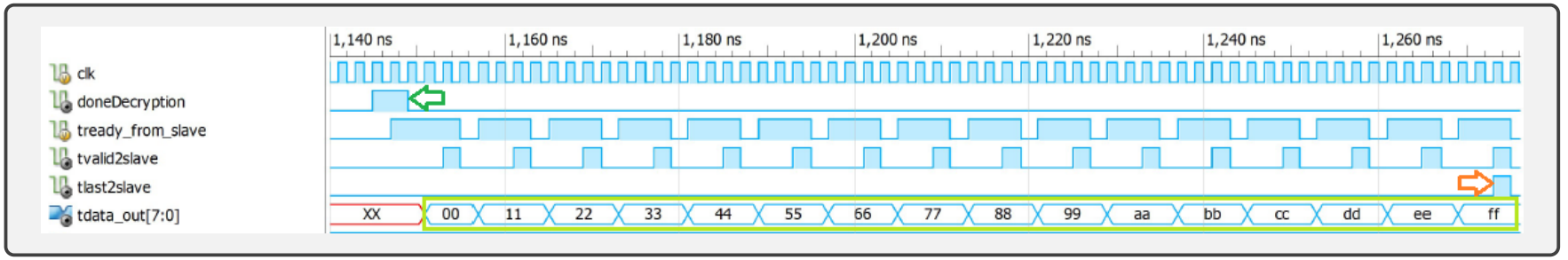
Figura 5.4: Simulación del Descifrador AES-128.



(a) Envío de Clave de 256-bit cada 32-bit



(b) Clave expandida y Dato enviado cada 8-bit



(c) Criptograma

Figura 5.5: Simulación del Cifrador AES-256.

Como se explicó en la Sección 4.2, se desarrollaron tres diseños hardware del algoritmo AES tanto para el cifrador como para el descifrador. Al simular dichos diseños se obtuvo información que permite conocer el rendimiento de cada uno de ellos, y con el cual es posible tomar desiciones sobre cual es más conveniente usar para alguna aplicación donde se quiera cifrar. Las transformaciones por si solas no utilizan ni el 1% de los recursos totales de la tarjeta (53200 Slice LUTs) (ver Tabla 5.2), en cambio el algoritmo AES en sus tres diseños nos permite observar más características. En la tabla 5.1 se enlistan las implementaciones hardware tanto del cifrador como del descifrador y la rutina de la clave, donde la columna *SliceLUTs* respresenta la cantidad y el porcentaje usado de conjuntos de "lookup tables", *SliceRegisters* representa la cantidad y el porcentaje de conjuntos de "registros", *Fmax* indica la frecuencia máxima del elemento (ciclos por segundo) dada en [MHz], *Latencia* indica cuantos ciclos de reloj ocurrieron desde que se procesó el primer dato de entrada hasta que fue devuelto el primer dato de salida, *Tiempo de Ejecución* indica el tiempo en segundos que tardó el elemento en procesar un bloque de datos, y finalmente la columna *Throughput* representa el número de bits por segundos que pueden ser procesados por el elemento.

Un diseño combinacional (Cipher/Decipher AES-comb) utiliza alrededor de un 17% de los *SliceLUTs* disponibles y a una baja frecuencia que oscila los 80 [MHz] pero que tiene una latencia de un ciclo de reloj, lo que significa que se obtiene una salida en un ciclo de reloj pero usando un número considerable de recursos, un uso de recursos parecido (17-19%) al del diseño pipeline (Cipher/Descipher AES-reg) donde la aquitectura de la que lleva el nombre permite disminuir el tiempo de ejecución y aumenta el Throughput de forma significativa, pues a pesar de que pasan 80 ciclos de reloj para obtener el primer dato de salida, los siguientes sólo tardan un ciclo de reloj. Finalmente, el diseño secuencial (Cipher/Decipher AES-fsm) trabaja con una frecuencia de 296.6 [MHz] más alta que la del diseño combinacional, además cuanta con un Throughput de 1.8 [Gbps] y de 2.9 [Gbps] para un AES-128 y un AES-256 respectivamente, y utiliza solo un 3-4% de recursos disponibles. Entonces, cada diseño tiene ventajas que pueden ser aprovechadas según se requiera. Por ejemplo, si se cuenta con recursos y el sistema no prioriza velocidad, pero si requiere una salida en un ciclo de reloj, el diseño combinacional sería la opción a elegir. Mientras que con una arquitectura pipeline se asegura un alto Throughput con un uso considerable de recursos, si eso se desea. El diseño que apenas utiliza un 3-4% de recursos del *Zynq* y que tiene un alto

Tabla 5.1: Rendimiento del IP Core AES-128 en sus tres implementaciones

ZedBoard Zynq XC7Z020	Element	Slice LUTs (of 53,200)		Slice Registers (of 106,400)		Fmax [MHz]	Latency [cycles] 128 / 256		Execution Time [ns] 128 / 256		Throughput [Mbits/s] 128 / 256	
Implementations	AES-comb	8,664	16%	-	-	81.29	1	-	12.3	-	10,406	-
	AES-reg	9,195	17%	5,248	4%	1,002.51	41	-	0.99	-	129,292	-
	AES-fsm	1,881	3%	415	0%	296.6	20	26	67.43	87.66	1,898	2,920
	Decipher AES-comb	9,925	18%	-	-	85.36	1	-	11.71	-	10,930	-
	Decipher AES-reg	10,433	19%	5,248	4%	631.791	80	-	1.58	-	81,012	-
	Decipher AES-fsm	1,774	3%	415	0%	285.27	20	26	70.1	91.14	1,825	2,808
Key Routine	KeyExpansion	121	0%	23	0%	278.56	455	466	1,592.5	1,631	80	156

Tabla 5.2: Recursos ocupados por cada Transformación descrita en hardware.

ZedBoard Zynq XC7Z020	Element	Slice LUTs (of 53,200)	
Initial & Final Registers	StateIn	1	0%
	StateOut	1	0%
Transformations	SuvBytes/invSB	640/512	1%
	ShiftRows/invSR	-	0%
	MixColumns/invMC	172/369	0%
	AddRoundKey	128	0%

Tabla 5.3: Comparación con Damjan et al. [37] y Algredo et al. [39]. Recursos.

ZedBoard Zynq XC7Z020 [37] / Virtex XC2V1000 [39]	Slices (of 5,120)	Slices LUTs (of 53,200) / Registers (of 106,400)		BRAM (of 40-XC2V1000 / 140-XC7Z020)
Damjan et al.[37] _Cipher	-	1,742	391	-
IP Core AES-fsm _Cipher $_{Zynq}$	-	1,881	415	0
Algredo et al. [39] _Cipher	586	-	-	10
IP Core AES-fsm _Cipher $_{Virtex}$	505	-	-	16
Damjan et al.[37] _Decipher	-	1,763	391	-
IP Core AES-fsm _Decipher $_{Zynq}$	-	1,744	415	0

Tabla 5.4: Comparación con Damjan et al. [37] y Algredo et al. [39]. Throughput.

ZedBoard Zynq XC7Z020 [37] / Virtex XC2V1000 [39]|	Fmax [MHz]	Throughput [Mbits/s] 128 / 256		[%] about [37] / [39]	
Damjan et al.[37] _Cipher	140	-	1,120	-	-
IP Core AES-fsm _Cipher $_{Zynq}$	296.6	1,898	2,920	+ 160 %	-
Algredo et al. [39] _Cipher	96.42	1,450	-	-	-
IP Core AES-fsm _Cipher $_{Virtex}$	194.8	1,246	1,917	-	- 14%
Damjan et al.[37] _Decipher	130	-	1,040	-	-
IP Core AES-fsm _Decipher $_{Zynq}$	373.27	2,388	3,675	+ 253%	-

Tabla 5.5: Comparación con Damjan et al. [37] a una frecuencia de 100 [MHz].

ZedBoard Zynq XC7Z020	Fmax [MHz]	Throughput [Mbits/s] 128 / 256		[%] about [37]
Damjan et al.[37] _Cipher	100	-	914	-
IP Core AES-fsm _Cipher	100	140.65	206.45	22.6 %
Damjan et al.[37] _Decipher	100	-	914	-
IP Core AES-fsm _Decipher	100	246.15	441.37	48 %

Throughput, como se mencionó en líneas anteriores, superior a los otros dos diseños fue el elegido para ser implementado en la *ZedBoard Zynq XC7Z020* debido a que se necesita use pocos recursos y tenga un rendimiento medio, debido a que será colocado en una cadena de comunicaciones con varios IP Cores propios de un Sistema de Telecomunicaciones que estarán compartiendo espacio en la misma tarjeta *ZedBoard*.

Por otra parte, se hizo una comparación de éste documento con la literatura, de donde se escogieron dos artículos para dicho fin (AES256 [37] y AES128[39]), y en la Tabla 5.4 se muestra la comparativa de la implementación secuencial del cifrador/descifrador de bloque **AES-128/256**. Las características mencionadas son la Frecuencia Máxima (Fmax [MHz]), Throughput ([Mbit/s]) y el porcentaje que referencia a ésta tesis con respecto a las referencias. La frecuencia máxima de la implementación del cifrador AES es de 296.6[MHz], 111% mayor en comparación con [37] y 102% mayor en comparación con [39], así como también el *Throughput* es 160%, a cambio de usar más recursos de los cuales usa 0.27% de *Slice LUTs* más que [37]. Y para poder realizar una comparación justa con [39], se sintetizó el cifrador para el dispositivo XC2V1000 donde se obtuvo 14% menor de Throughput con respecto a ésta tesis y un uso de recursos similar pues sólo se usaron 81 Slices menos y 6 BRAM más que [39] (ver Tabla 5.3).

También se hizo el cálculo para una frecuencia máxima de 100 [MHz] tal cual lo hizo [37], donde ésta tesis tiene sólo el 22.6 % y 52 % de *Throughput* para el cifrador y descifrador respectivamente, que la referencia [37].

En la siguiente sección se describe la implementación del cifrador y del descifrador *AES 128/256* que hasta ahora ha sido simulado. El ejemplo mostrado corresponde a un cifrador AES-128, sin embargo, la arquitectura hardware es la misma si se escogiera un ejemplo para un cifrador AES-256.

5.2 Implementación

Una vez que se cuenta con los IP Cores AES de cifrado y descifrado, el siguiente paso es su implementación en la tarjeta de desarollo *ZedBoard*. Entonces, para hacer la demostración de cifrar un dato y descifrarlo después, se conectan uno tras de otro mediante su interfaz *AXI4-Stream* como se observa en la Fig. 5.6.

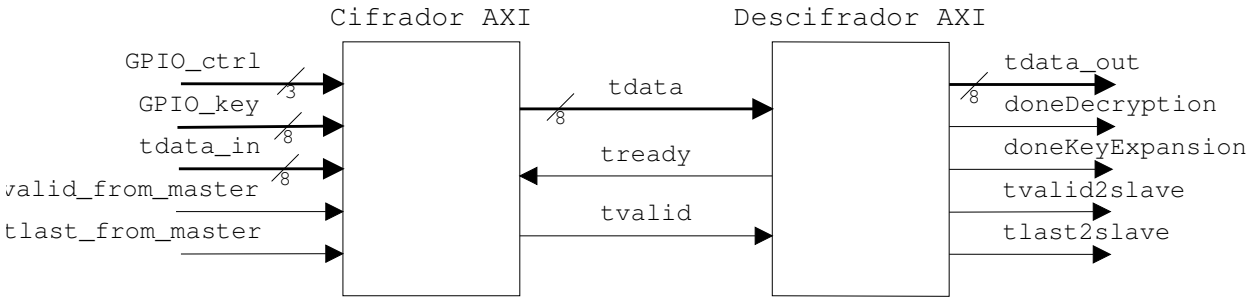


Figura 5.6: Conexión AXI del Cifrador con el Descifrador

El elemento que funge como maestro y esclavo a la vez en la conexión con los IP Cores AES es el manejador de memoria AXI_DMA (AXI Direct Memory Access) que se encarga de accesar directamente a memoria para leer los datos que serán procesados por el Cifrador AXI, y de guardar los datos resultantes obtenidos del Descifrador AXI.

El diagrama de bloques del sistema de cifrado y descifrado que se implementa en la tarjeta de desarrollo *ZedBoard* es mostrado en la Fig. 5.7.

Los elementos que forman parte del sistema son:

AXI Direct Memory Access. Core AXI que provee acceso directo a memoria entre la memoria asignada para los datos de origen y las interfaces AXI. Los puertos utilizados son MM2S (AXI4 Stream Master) y S2MM(AXI4-Stream Slave), ambos independientes.

ZYNQ Pocessing System. Es un SoC que contiene un sistema de procesamiento(PS) y una unidad lógica programable(PL). Del *PS* se usa UART para configurar mediante comunicación serial el sistema de Cifrado-Descifrado e iniciar la transferencia de información, y de *PL* un AXI GPIO.

aes128bits_AXI. AXI IP Core AES de cifrado de datos.

descifrador_AXI. AXI IP Core AES de descifrado de datos.

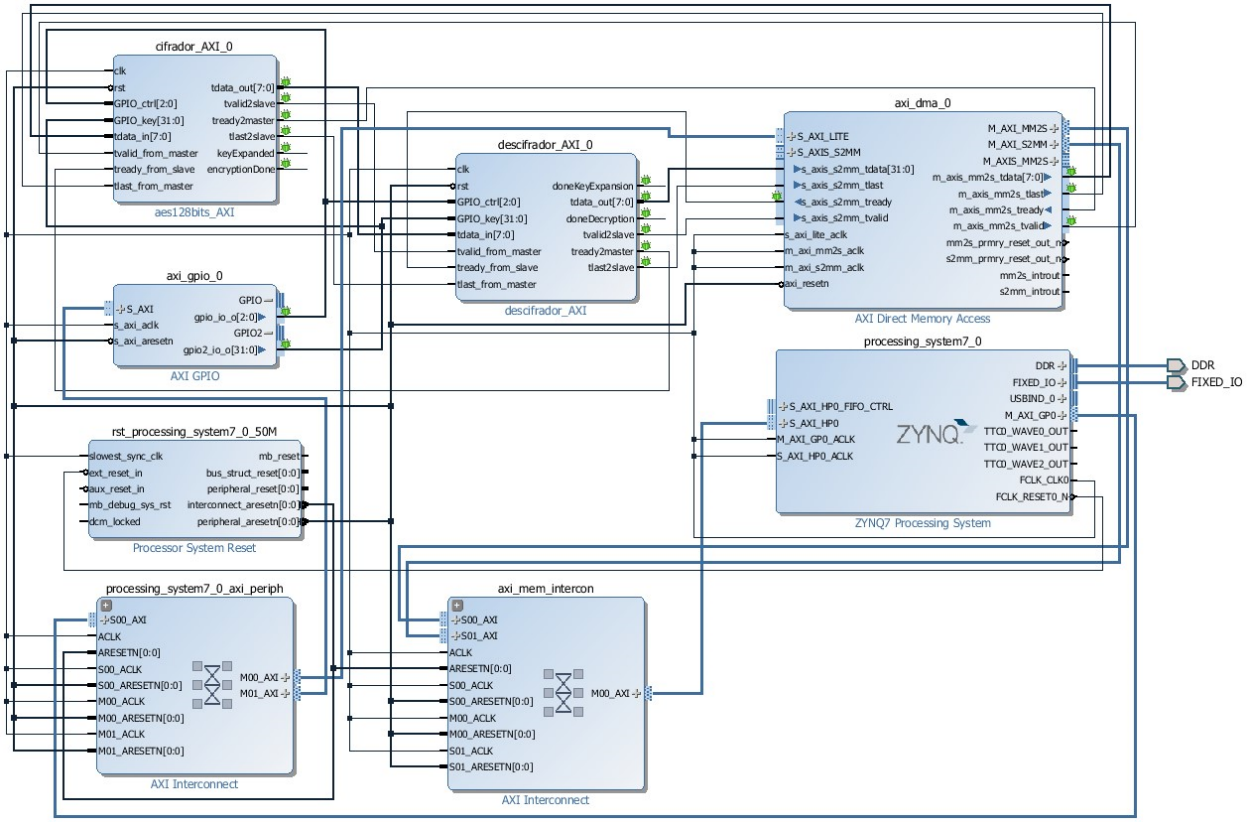


Figura 5.7: Digrama de bloques de una cadena de Cifrado-Descifrado AES-AXI4 Stream en la plataforma Vivado

AXI GPIO. AXI IP Core de propósito general, por lo que puede ser usado como entrada o salida. En éste caso, se utilizó un AXI GPIO con sus dos puertos activados como salidas, una que contiene a la señal *GPIO_ctrl* y otra a *GPIO_key*.

Processor System Reset. IP Core que provee de *resets* basados en las condiciones de reinicio externo e interno. Los dos resets usados son activos en bajo, donde uno alimenta a los periféricos y el otro a los AXI Interconnect.

AXI Interconnect. IP Core que conecta a unidades maestro AXI a unidades esclavo AXI. En éste caso, conecta el *AXI GPIO* con el *Zynq7 Processing System*, y el *AXI DMA* con el *Zynq7 Processing System*.

Existe también un entorno de Vivado que involucra la interacción con el hardware por medio del lenguaje de programación C, dicho entorno es llamado *SDK*. La aplicación permite teclear

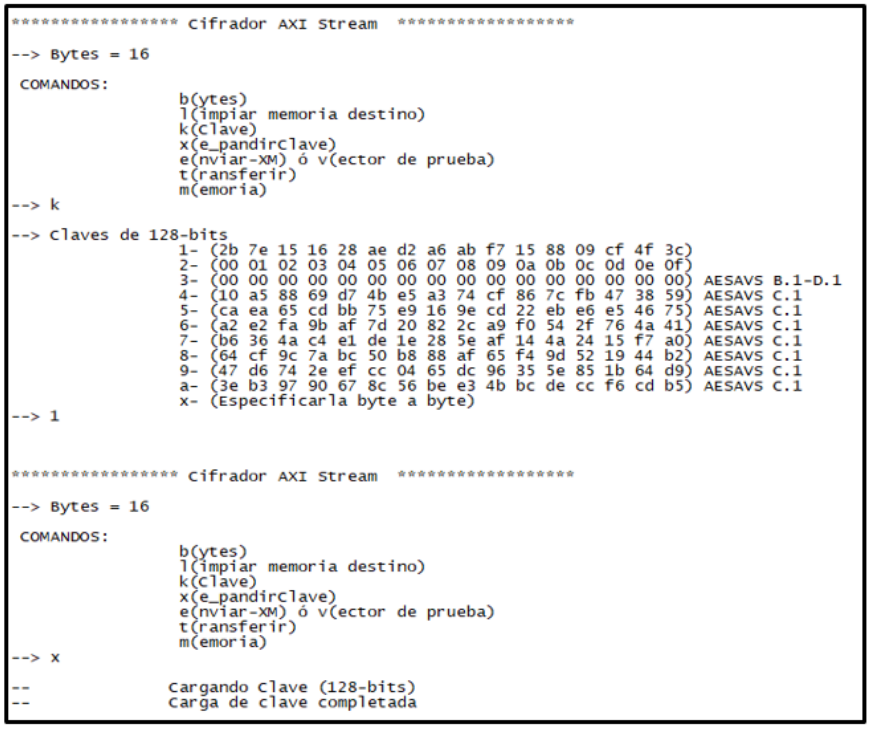
comandos y se lleve a cabo el cifrado y descifrado AES hardware, donde los comandos que son principalmente utilizados son:

(k) Comando que muestra las claves de prueba (128-bit) a usar, o también permite especificar una diferente con el subcomando x.
(x) Comando que después de haber escogido la clave, ésta es expandida.
(v) Comando que muestra los vectores de prueba (128-bit) a cifrar, o también permite especificar un dato diferente con el subcomando x.
(t) Comando que inicia la transferencia del AXI DMA que lee el vector guardado en memoria y lo envía byte a byte al cifrador_AXI_0 para posteriormente guardar el vector generado por descifrador_AXI_0 en otra sección de memoria.

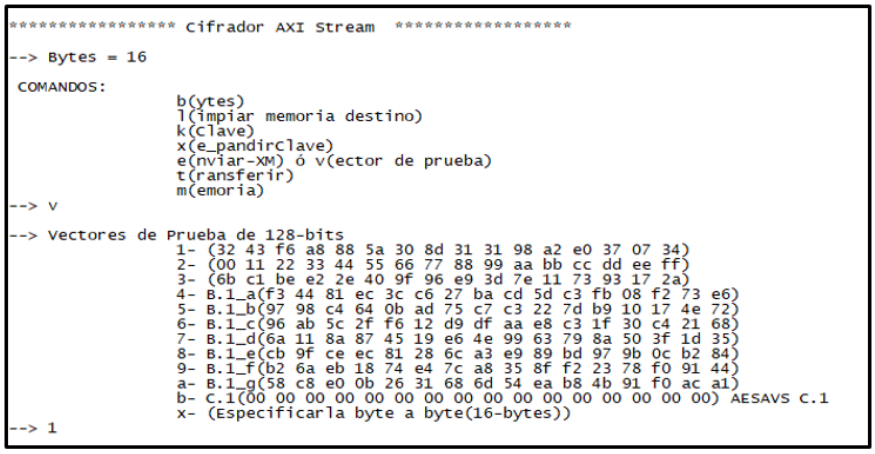
Ahora, con la descripción de los elementos involucrados en la implementación hardware, es posible describir la interacción de los mismos en el proceso de cifrado-descifrado:

a. El AXI GPIO de doble puerto conecta las señales *GPIO_ctrl* y la señal *GPIO_key* a los dos Cores AES, señales que contienen el tipo de AES a utilizar y la clave privada del algoritmo respectivamente.
b. El dato que se quiere cifrar estará guardado en memoria y el DMA se hará cargo de leer byte a byte hasta completar el bloque de 128-bit necesarios por el cifrador_AXI_0.
c. Para poder iniciar la transferencia de los datos guardados en memoria al cifrador_AXI_0 y posteriormente guardar los datos arrojados por descifrador_AXI_0 en memoria, se necesita expandir la clave.
d. Se teclea k para seleccionar la clave, que para el caso ejemplo, se selecciona la clave #1. (ver Fig. 5.8a)
e. Se teclea x para que la clave seleccionada sea expandida.
f. Se teclea v para seleccionar el dato de 128-bit que será cifrado. Para el caso ejemplo se selecciona el vector #1. (ver Fig. 5.8b)
g. Se teclea t para iniciar la transferencia.

Para cuando el proceso ha terminado, el dato de entrada inicial y el dato de entrada final debe ser el mismo, puesto que el criptograma resultante del elemento cifrador_AXI_0 será reconstruido por el elemento descifrador_AXI_0 a su forma original.



(a) Comando k y x



(b) Comando v

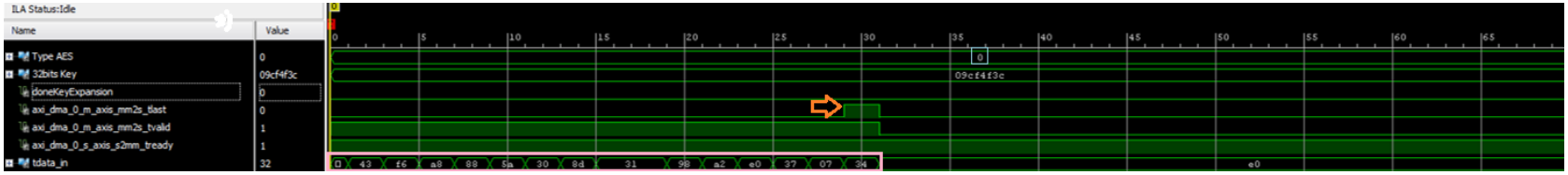
Figura 5.8: Interfaz Software-Hardware : Datos de entrada

La manera de verificar que el proceso haya terminado de la forma esperada, es haciendo uso del Core ChipScope que es un analizador lógico personalizable que puede monitorear cualquier señal del sistema hecho en Vivado. Para indicar las señales a ser monitoreadas, se seleccionan con el botón derecho del ratón y se da click en la opción `Mark Debug`. Siguiendo con el ejemplo, se monitoriean todas las señales de entrada del `cifrador_AXI_0` y todas las señales de salida del mismo, una de ellas es el criptograma que a su vez es la entrada del `descifrador_AXI_0` (*tdata_out* –> *tdata_in*). También se monitorean las demás entradas y salidas del `descifrador_AXI_0`, donde la salida *tdata_out* contiene el criptograma descifrado.

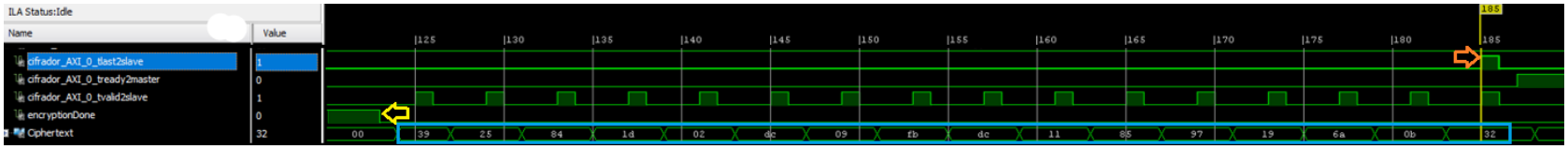
En la Fig. 5.9 se muestra el comportamiento del IP Core AES de cifrado encadenado al IP Core AES de descifrado directamente en la tarjeta, pues con la herramienta antes descrita, es posible seguir las señales que intervienen en la transferencia de la información desde el DMA hasta su regreso al mismo. Una vez que se ha escogido la clave y ha sido expandida, al teclear el comando `t`, se genera el diagrama que contiene las señales seleccionadas para ser monitorieadas y su correspondiente valor. Como es sabido, cuando la señal del DMA *tlast* tiene el valor de 1 (marcada con un flecha color naranja ver Fig. 5.9a), ha terminado de enviar el último byte al Core `cifrador_AXI_0` y para cuando éste último haya terminado de procesar el bloque de 16-byte activará la señal *encryptionDone* (marcada con un flecha color amarillo, ver Fig. 5.9b),y mientras la señal del `descifrador_AXI_0` *tvalid* sea 1, le comenzará a enviar el criptograma byte a byte activando la señal *tlast* cuando haya terminado el bloque (enmarcados en color azul).

Finalmente, el `descifrador_AXI_0` procesará el criptograma para recuperar el dato original activando la señal en 1 *doneDecryption* (marcada con una flecha color azul, ver Fig. 5.9c). Después, mientras la señal *tvalid* del puerto esclavo del DMA sea 1, le será enviado byte a byte el bloque descifrado (enmarcado en color amarillo), y de igual modo activará *tlast* para indicar que el último byte ha sido enviado (señal marcada con una flecha color naranja). Con el procedimiento anterior se verifica el proceso y los datos resultantes de cada IP core AES-128, pues directamente en la tarjera con el *ChipScope* fueron comparados los datos resultantes con los del ejemplo presentado en el apéndice B de [17].

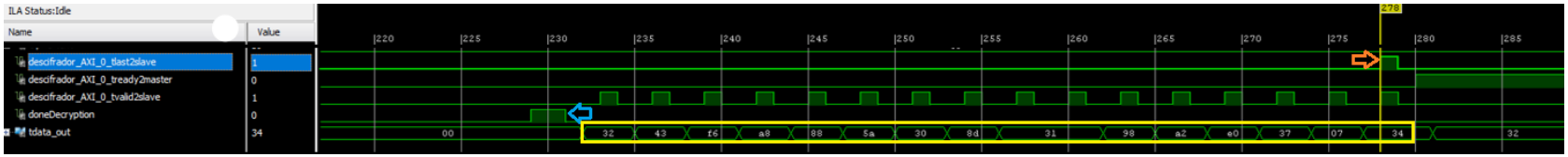
Dado que la implementación hardware del cifrador AES se llevó a cabo y se pudo mostrar en tiempo real del hardware mediante la herramienta *ChipScope*, se continua con las conclusiones del desarrollo del IP Core AES con interfaz AXI.



(a) Carga del bloque 128-bit



(b) Fin de Cifrado - Descifrador recibe bloque cifrado 128-bit



(c) Bloque original 128-bit recuperado

Figura 5.9: Proceso Cifrado-Descifrado AES-128.

Conclusiones

En este trabajo de tesis, se simuló y se implementó un algoritmo de cifrado de bloques simétrico AES descrito en *Verilog*, con una interfaz estándar AMBA reconocida (AXI4-Stream) que le permite comunicarse con otros IP Cores AXI en la tarjeta de desarrollo *ZedBoard*™. De este modo, pudo ser incluido en una cadena que comprendía al cifrador y al descifrador y con la cual se verificó la comunicación AXI entre los IP Cores de cifrado AES y otros Cores AXI. Así mismo, se tiene una interfaz de software que permite la interacción con el hardware, dicha interfaz es requerida para la implementación del sistema de cifrado-descifrado, es decir, se necesita interactuar con la arquitectura hardware si se quiere realizar una configuración inicial (por ejemplo: cambiar la clave, cambiar el tipo de AES y por ende la clave, así como para iniciar la transferencia AXI).

El desempeño del IP Core AES es tal, que se muestra un mínimo consumo de recursos y en un tiempo de ejecución por bloque de 16-byte de 67.43 [ns] y un rendimiento de 1.8 [Gbps] y 2.9 [Gbps] para un AES-128/256 respectivamente, haciéndolo candidato para su aplicación final en una cadena de Telecomunicaciones en la que dichas características tienen gran importancia. Además, en caso de que el IP Core AES fuera utilizado en otras aplicaciones con interfaz *AXI4-Stream*, se tienen tres opciones a usar dependiendo de las necesidades de la misma.

El IP Core AES está desarrollado en su modo natural que es el modo *ECB* y se encuentra preparado para configurar el tamaño de clave a 128-bit y a 256-bit. En tanto que una de las ventajas de utilizar el lenguaje *HDL Verilog* es el salto casi inmediato para poder verificar hardware en *SystemVerilog*.

En la revisión del estado de la cuestión, se muestran varias implementaciones hardware de AES donde sólo se encontraron dos referencias que desarrollan un algoritmo de cifrado AES, una con interfaz estándar ([37]) que le permita incorporarse a un sistema determinado, ventaja que permite a un IP Core AES AXI4-Stream tener varias aplicaciones hardware en entornos que en su

mayoría sean AXI4-Stream. Y una implementación mexicana ([39]) que tiene un rendimiento de 1.45 [Gbps], resultados cercanos a éste trabajo siendo que el diseño se implementó para la tarjeta Virtx utilizada en dicha referencia. En cuanto al uso de recursos, las dos referencias hacen uso de bloques RAM (BRAMs) para las tablas de sustitución, por lo que usan menos *Slices LUTs* y *Slices Registers*. Ésta tesis no hace uso de BRAMs y es por ello que el uso de *Slices LUTs* y *Slices Registers* se incrementa, hecho que beneficia en cuanto a portabilidad, pues el HDL funciona sin importar que Xilinx no lo use.

Como trabajo futuro del proyecto está el desarrollar el IP Core AES en sus varios modos de funcionamiento en los cuales sólo varía la forma en que es alimentado el Core, puesto que el algoritmo AES continúa siendo el mismo. También se puede agregar la implementación del tamaño de clave 192-bit para el cual se necesita el desarrollo de un elemento hardware que calcule la operación módulo para múltiplos de 6.

Referencias

[1] L. E. Bassham III, "The advanced encryption standard algorithm validation suite (AESAVS)", *NIST Information Technology Laboratory(2002)*.

[2] P. Brey, "Ethical aspects of information security and privacy", *Security, privacy, and trust in modern data management, Springer Berlin Heidelberg*, pages 21-36,2007.

[3] CISCO, "2017 Annual Cybersecurity Report", (2017), [Online] http://b2me.cisco.com/en-us-annual-cybersecurity-report-2017.

[4] P. Kenellis, E. Kiountouzis, N. Kololotronics and D. Martakos, "Digital Crime and Forensic Science in Cyberspace", I. G. Inc., 357pp., page 17, 2006.

[5] CISCO, "Cisco 2017 Annual Cybersecurity Report Graphics", (2017), [Online] https://www.cisco.com/c/en/us/products/security/security-images-acr2017.html?_ga=2.107486981.68724943.1507142350-589790453.1507142350.

[6] M. B. Gokhale and P. S. Graham, "Reconfigurable computing: Accelerating computation with field-programmable gate arrays", Springer, 2006.

[7] L. Ponemon, "Know the Odds: The Cost of a Data Breach in 2017", (Julio, 2017), [Online] https://securityintelligence.com/know-the-odds-the-cost-of-a-data-breach-in-2017/.

[8] M. Rouse, Definition record, TechTarget-SearchOracle, (2005), [Online] http://searchoracle.techtarget.com/definition/record

[9] S. M. Trimberger,"Three ages of FPGAs: A retrospective on the first thirty years of FPGA technology",*Proceedings of the IEEE (2015)*, Volume 103, No. 3, pages 318-331.

[10] M. Dworkin, "Recommendation for block cipher modes of operation. Methods and techniques", NATIONAL INST OF STANDARDS AND TECHNOLOGY GAITHERSBURG MD COMPUTER SECURITY DIV, 2001.

[11] J.G. Proakis and M. Salehi, "Communication System Engineering", Prentice Hall, 2nd Ed., 794 pp., 2002.

[12] J.G. Proakis and M. Salehi, “Fundamentals of Communication System”, Pearson, 2nd Ed., 903 pp., 2005.

[13] W. Diffie, M.E. Hellman, “New Directions in Cryptography”,*IEEE Transaction on Information Theory(1976)*,Volume 22,pages 644-654.

[14] R. Housley,Cryptographic Message Syntax, (2009), [Online] https://tools.ietf.org/html/rfc5652.

[15] J. Gutiérrez, J.G. Tena, “Protocolos Criptográficos y Seguridad en Redes”, Ed. Universidad de Cantabria,2003, pages 46-50.

[16] J. Daemen and V. Rijmen, “The Design of Rijndael. AES The Advanced Encryption Standard”, Springer-Verlag, 238 pp., 2001.

[17] NIST, “Advance Encryption Standard (AES)”, Federal Information, Processing Standards Publication 197, 47 pp., November 26 2001.

[18] L. H. Crockett, R. A. Elliot, M. A. Enderwitz and R. W. Stewart, “The Zynq Book: Embedded Processing with the Arm Cortex-A9 on the Xilinx Zynq-7000 All Programmable Soc”, Strathclyde Academic Media, 2014.

[19] O. S. Dhede, S.K. Shah *et al*, “A Review: Hardware Implementation of AES Using Minimal Resources on FPGA”, *International Conference on Pervasive Computing (ICPC) 3 pp.*, 2015.

[20] Ming-Haw Jing, Zih-Heng Chen, Jian-Hong Chen, Yan-Haw Chen, “Reconfigurable system for high-speed and diversified AES using FPGA”, *Microprocessors and Microsystem, ELSEVIER, Vol 31*, pages 94-102,2007.

[21] V. Venugopal and D.M. Shila, “High throughput implementations of cryptography algorithms on GPU and FPGA”, *International Conference on Instrumentation and Measurement Technology Conference(I2MTC), IEEE*, pages 723-727,2013.

[22] S. R. Huddar, S. R. Rupanagudi, R. Ravi, S. Yadav and S. Jain 2013, “Novel architecture for inverse mix columns for AES using ancient Vedic Mathematics on FPGA Advances in Computing”, *Communications and Informatics (ICACCI), International Conference on IEEE*, 2013.

[23] Aysu, C. Patterson and P. Schaumont, “Low-cost and area-efficient FPGA implementations of lattice-based cryptography Hardware-Oriented Security and Trust (HOST)”, *International Symposium on IEEE*, p. 81-86, 2013.

[24] Liu, L. Ju, X. Cai, Z. Jia and Z. Zhang, “High performance FPGA implementation of elliptic curve cryptography over binary fields”, *Trust, Security and Privacy in Computing and Communications (TrustCom), 2014 IEEE 13th International Conference on IEEE* p. 148-155, 2014.

[25] P. Swierczynski, M. Fyrbiak, P. Koppe and C. Paar, "FPGA Trojans through Detecting and Weakening of Cryptographic Primitives", *Transactions on Computer-Aided Design of Integrated Circuits and Systems, IEEE, Vol 34*, pages 1236-1249, 2015.

[26] L.S. Abhiram, B.K. Sriroop, L. Gowrav, K.H.L. Punith, M.C. Lakkannavar *et al*, "FPGA implementation of dual key based AES encryption with key Based S-Box generation", *International Conference on Computing for Sustainable Global Development(INDIACom)*, pages 577-581, 2015.

[27] Xilinx, "AXI Reference Guide", UG761 (v13. 1), (2011), [Online] http://www.xilinx.com/support/documentation/ipdocumentation/ug761.

[28] Paar and J. Pelzl, "Understanding cryptography: a textbook for students and practitioners", Springer Science & Business Media, 372 pp., 2009.

[29] D. M. Burton, "Elementary number theory", Tata McGraw-Hill Education, 2006.

[30] I. Camacho, "Desarrollo en Verilog de librería de núcleos básicos de hardware (IP Cores) para aplicaciones de alto rendimiento", Tesis UAZ, 2016.

[31] E. F. Foundation, "EFF DES Cracker Machine Brinsgs Honesty to Crypto Debate. Electronic Foundation Proves that DES is not secure ", (1998), [online] https://w2.eff.org/Privacy/Crypto/Crypto_misc/DESCracker/HTML/19980716_eff_descracker_pressrel.html (2017).

[32] A. J. Menezes, P. C. Van Oorschot and S. A. Vanstone, "Handbook of applied cryptography", CRC press,779 pp., 1996.

[33] P. Swierczynski, M. Fyrbiak, P. Koppe and C. Paar, "FPGA Trojans through detecting and weakening of cryptographic primitives", IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, p. 1236-1249, 2015.

[34] A. Ibrahim, "FPGA-based Hardware Implementation of Compact AES Encryption Hardware Core", WSEAS Transactions on Circuits and Systems, No. 14, p. 365-372, 2015.

[35] Xie, P. K. Meher and Z.-H. Mao, "High-throughput finite field multipliers using redundant basis for FPGA and ASIC implementations", IEEE Transactions on Circuits and Systems I: Regular Papers, p. 110-119, 2015.

[36] C. Beierle, J. Jean, S. Kölbl, G. Leander, A. Moradi, T. Peyrin, Y. Sasaki, P. Sasdrich and S. M. Sim, "The SKINNY Family of Block Ciphers", *NIST (2016)*.

[37] D. Rakanovi and R. Struharik, "IP core for AES256 and TDES algorithms with AXI interface", *Telecommunications Forum (TELFOR), 24th IEEE*, p. 1-4, 2016.

[38] B. Chellam and R. Natarajan, "AES Hardware Accelerator on FPGA with Improved Throughput and Resource Efficiency Arabian Journal for Science and Engineering, p. 1-18, 2017.

[39] I. Algredo-Badillo, C. Feregrino-Uribe and R. Cumplido, “Design and implementation of an FPGA-Based 1.452-gbps non-pipelined AES architecture”, Computational Science and Its Applications-ICCSA, p. 456-465, 2006.

[40] D. K. V. Morales, “Encriptador de señal mediante osciladores caóticos implementados en FPGA”, 2016.

[41] I. Algredo-Badillo, R. A. Cumplido-Parra, and C. Feregrino-Uribe, “Desarrollo de un Módulo MD5 para un Sistema Criptográfico Reconfigurable en un FPGA”, por aparecer en International Conference on Reconfigurable Computing and FPGAs, ReConFig04, México, 2004.